VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE MIKROSKOPICKÉ SIMULACE DOPRAVY ZA POUŽITÍ OPENCL

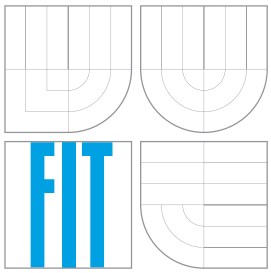
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

ANDREJ URMINSKÝ

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE MIKROSKOPICKÉ SIMULACE DOPRAVY ZA POUŽITÍ OPENCL

ACCELERATION OF MICROSCOPIC URBAN TRAFFIC SIMULATION USING OPENCL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — ANDREJ URMINSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVOL KORČEK
SUPERVISOR

BRNO 2011

Abstrakt

S narastajúcim počtom vozidiel na našich cestách sa čoraz väčšmi stretávame so súčasnými problémami dopravy, medzi ktoré by sme mohli zaradiť početnejšie havárie, zápchy a zvýšenie vypúšťaných emisií CO2, ktoré znečisťujú životné prostredie. Na to, aby sme boli schopní efektívne využívať cestnú infraštruktúru, nám môžu poslúžiť napríklad simulátory dopravy. Pomocou takýchto simulátorov môžme vyhodnotiť vývoj premávky za rôznych počiatočných podmienok a tým vedieť, ako sa správať a reagovať v rôznych situáciách dopravy. Táto práca sa zaoberá témou akcelerácia mikroskopickej simulácie dopravy za použitia OpenCL. Akcelerácia simulácie je dôležitá pri potrebe analyzovať veľkú sieť infraštruktúry, kde nám bežné spôsoby implementácie simulátorov nestačia. Pre tento účel je možné použiť napríklad techniku GPGPU súčasných grafických kariet, ktoré sú schopné paralelne vykonávať všeobecné výpočty. Pri tvorbe tejto práce bola použitá práve táto technika pre urýchlenie výpočtov a boli dosiahnuté niekoľkonásobné zrýchlenia na GPU oproti paralelnej implementácii na CPU.

Abstract

As the number of vehicles on our roads increases, the problems related to this phenomenon emerge more dramatically. These problems include car accidents, congestions and CO2 emissions production, increasing CO2 concentrations in the atmosphere. In order to minimize these impacts and to use the road infrastructure effectively, the use of traffic simulators can come in handy. Thanks to these tools, it is possible to evaluate the evolution of a traffic flow with various initial states of the simulation and thus know what to do and how to react in different states of the real-world traffic situations. This thesis deals with acceleration of microscopic urban traffic simulation using OpenCL. Supposing it is necessary to simulate a large network traffic, the need to accelerate the simulation is necessary. For this purpose, it is possible, for example, to use the graphics processing units (GPUs) and the technique of GPGPU for general purpose computations, which is used in this work. The results show that the performance gains of GPUs are significant compared to a parallel implementation on CPU.

Klíčová slova

mikrosimulácia dopravy, celulárne automaty, GPGPU, OpenCL

Keywords

traffic microsimulation, cellular automata, GPGPU, OpenCL

Citace

Andrej Urminský: Akcelerace mikroskopické simulace dopravy za použití OpenCL, diplomová práce, Brno, FIT VUT v Brně, 2011

Akcelerace mikroskopické simulace dopravy za použití OpenCL

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Pavla Korčeka.

.....
Andrej Urminský
24. května 2011

Poděkování

Rád by som poďakoval Ing. Pavlovi Korčekovi za odbornú pomoc, ktorú mi ochotne poskytol pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

V každodennom živote sa stretávame so súčasnými problémami automobilovej dopravy, medzi ktoré patria napr. zápchy, vysoká a narastajúca ročná celosvetová spotreba paliva (ropy, zemného plynu...), znečisťovanie ovzdušia a životného prostredia, v ktorom žijeme atp. Na automobilovej preprave sme závislí, využívame ju pre prepravu osobnú, nákladnú. Z roka na rok jej využívanie vďaka lepšej dostupnosti a cenám narastá a je teda nutné sa týmito problémami zaoberať.

1.1 Vývoj dopravy a jej súčasný stav

Vývoj automobilovej dopravy v Českej republike a na Slovensku má podobný priebeh [32], [51]. Zaujímavé sú napríklad štatistiky Ministerstva dopravy Slovenskej republiky, ktoré ukazujú, že za posledných 14 rokov od roku 1995 do roku 2009 sa počet prepravených osôb verejnou dopravou (mestská hromadná, cestná verejná, železničná, ...) znížil skoro na polovicu [32]. A naopak, skoro celá táto strata prepravených osôb verejnou dopravou sa presunula do individuálnej automobilovej dopravy. V roku 1995 to bolo vyše 1,3 mld osôb a v roku 2009 už vyše 1,8 mld osôb prepravených individuálnou automobilovou dopravou [32]. Podiel súkromnej a verejnej dopravy sa v percentuálnom vyjadrení v roku 1995 pohyboval na úrovni skoro 50:50, dnes je to už skoro 70:30.

Ako nám už tieto čísla napovedajú, musel takisto vzrásť aj počet registrovaných osobných automobilov, a to za 14 rokov o viac ako 50% z niečo vyše milióna, na skoro 1,6 milióna áut v roku 2009 [31]. Tu narážame na problém, pretože, čo sa výstavby infraštruktúry týka, väčšina krajín, vrátane Slovenska a Čiech, nie je schopná s takýmto tempom súperiť [30] a teda aj hustota premávky na cestách rastie.

S vyššou hustotou premávky sa nám zvyšuje pravdepodobnosť havárií. Takisto tomu prispieva aj nervozita šoférov a ich potreba častejšie predbiehať na cestách I., II. a III. triedy.

Plynulosť jazdy je narušená. Je nutné častejšie brzdiť a pridávať, čím sa zase zvyšuje spotreba áut. S vyššou spotrebou prichádza aj už spomínané znečisťovanie životného prostredia.

1.2 Spôsoby riešenia problémov v automobilovej doprave

Máme niekoľko spôsobov, akými sa dajú tieto problémy do určitej, ale význačnej miery potlačiť. Aby však bolo naozaj možné dospieť k významným výsledkom obecne, nie je

postačujúce zamerať sa iba na jeden spôsob riešenia, ale je nutná ich korelácia. Medzi najdôležitejšie spôsoby patria [52]:

- **Nové, efektívnejšie a úspornejšie technológie** - Výrobcovia automobilov prichádzajú s technológiami, ako znížiť spotrebu paliva a emisií CO2. Napríklad pomocou 8-stupňových prevodoviek [20] alebo systémov, ktoré automaticky vypnú motor pri zastavení vozidla [50]. Medzi ďalšie technologické pokroky patria autá s hybridným pohonom, ktoré dokážu zredukovať emisie o 25-90% [17].

- **Osveta obyvateľstva** - Napriek tomu, že je ťažké byť v dnešnej dobe zbehlý a vzdelaný vo všetkých, príp. mnohých oblastiach, mali by sme vedieť dôležité aspekty nášho každodenného fungovania a života. Samozrejme sem patrí aj dopad našich činov a správania sa na životné prostredie. To môžme chrániť napríklad uprednostnením mestskej hromadnej dopravy pred osobným automobilom, pretože takýto spôsob dopravy môže byť lacnejší, efektívnejší, ekologickejší a v mnohých prípadoch aj rýchlejší [48].

  Ekonomickým šoférovaním vieme takisto ušetriť palivo a teda aj znížiť množstvo vypúšťaných emisií CO2. Medzi základné princípy takéhoto šoférovania patrí napr. plynulá jazda, používanie správne nahustených pneumatík, radenie do optimálnych prevodových stupňov, používanie klimatizácie iba v nevyhnutných prípadoch atď. Efektívnou metódou je používanie bicykla, prípadne ísť pešo. Približne 50% všetkých výjazdov autom je do okruhu 5km [19].

- **Modernizácia a výstavba nových ciest** - Veľkým prínosom pre súčasný stav automobilovej dopravy by bola výstavba potrebnej infraštruktúry. Tá je však veľmi drahá a obzvlášť v mestách je často z priestorových dôvodov neprijateľná.

  Ďalším možným prínosom je vývoj inteligentných zariadení a simulátorov, ktoré sú schopné simulovať a analyzovať situáciu na cestách za rôznych podmienok pravdepodobnosti príchodu áut, tvaru ciest, počtu pruhov na ceste atď. Takéto silné simulačné nástroje nám môžu pomôcť pochopiť a analyzovať navzájom poprepájané dynamické systémy a podporiť tak rozhodovanie pre lepšie plánovanie, monitorovanie a správne zvládanie poruchových situácií.

1.3 Cieľ a štruktúra práce

Táto práca sa zaoberá témou urýchlenia mikroskopickej simulácie dopravy pomocou technológie OpenCL. Prvá časť je zameraná na teoretický popis simulátorov pre riešenie problémov automobilovej dopravy a spôsobmi ich implementácie, medzi ktoré patria celulárne automaty, diskrétne paralelné výpočtové modely s lokálnou interakciou výpočtových elementov. Taktiež popisuje techniku paralelných výpočtov na grafických kartách (GPU) pomocou GPGPU.

Po teoretickej časti práce je popísaný spôsob návrhu a implementácie aplikácie spolu s testami aplikácie a porovnaním výsledkov testov.

Štruktúra práce je nasledovná: Kapitola 1 pojednáva o nežiadúcich vplyvoch súčasného stavu automobilovej dopravy a možných spôsoboch riešenia týchto problémov. Časť 2 slúži na popis základného paralelného výpočtového systému, ktorý bude použitý pre implementáciu simulátora. V ďalšej časti 3 sú popísané niektoré známe mikroskopické simulátory, na základe ktorých bude postavený aj simulátor dopravy navrhnutý v ďalšom pokračovaní

tejto práce. Takisto sú tu čitateľovi ukázané niektoré možné spôsoby, akými sa dajú urýchliť simulátory dopravy oproti simuláciám bez použitia špeciálnych techník a optimalizácií. S tým súvisia aj nasledujúce dve kapitoly, 4 a 5 hovoriace o vývoji architektúry grafických kariet a možnostiach implementácie metódy GPGPU na multiplatformných systémoch.

Kapitola 6 sa zaoberá popisom, akým spôsobom bola aplikácia navrhnutá a na akých zariadeniach testovaná. Ďalšie dve časti vysvetľujú, aká bola použitá dátová štruktúra programu (7) a spôsob, ako bol simulátor implementovaný (8). V dosiahnutých výsledkoch (9) je urobený rozbor testov prevádzaných nad jednotlivými časťami implementácie rôznych výpočtových zariadení (CPU a GPU). O problémoch pri implementácii hovorí kapitola 10 a tiež o ďalších možných rozšíreniach a budúcom vývoji v (11).

Kapitola 2

Celulárne automaty

Celulárny automat (CA) je diskrétny paralelný výpočtový model s lokálnou interakciou výpočtových elementov [11], [8]. Celulárne automaty sa dajú využiť ako matematické modely pre fyzikálne, biologické a počítačové systémy. Ich aplikácia má široké uplatnenie, napr. generovanie pseudonáhodných čísel, simulácia rastu kryštálov, simulácia systémov s lokálnymi interakciami (požiar lesa, doprava, ...), difúzia tepla a znečistenia, modely v biológii a ekológii, grafika (textúry, rozpoznávanie, ...) a mnoho ďalších. Majú jednoduchú konštrukciu a teda dostupný potenciál k presným matematickým analýzam s možnosťou zložitého správania.

V mikrosimuláciách s jednotlivými entitami, ktoré medzi sebou interagujú, je bežné, že časom nastane chvíľa, keď sa zo zmätku stáva poriadok a objavuje sa niečo nové: vzor, rozhodnutie, štruktúra alebo zmena smeru. Celulárne automaty vykazujú známky takéhoto emergentného správania a samo-organizácie.

2.1 Princíp celulárnych automatov

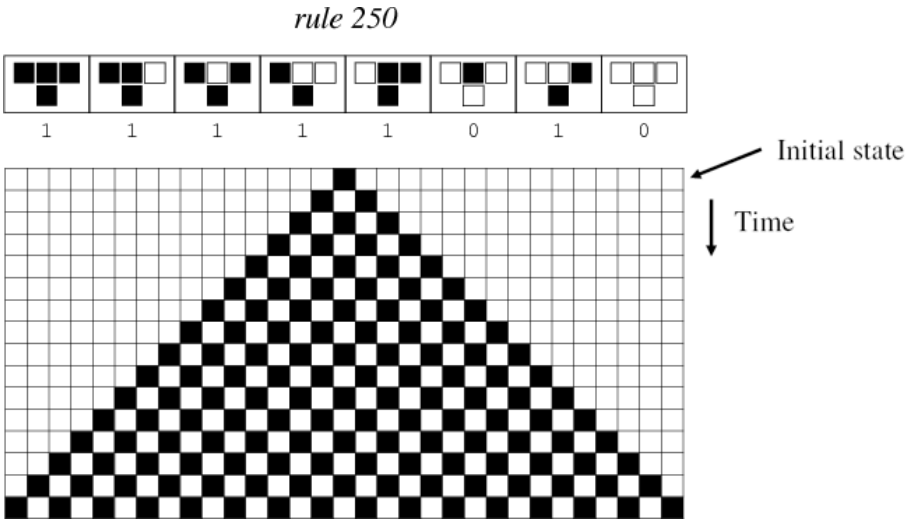
Jednoduchý celulárny automat pozostáva z elementov - buniek, ktoré sú obvykle usporiadané do pravidelnej mriežky. Každá bunka sa môže nachádzať v jednom z konečnej množiny stavov. Každá bunka obsahuje lokálnu prechodovú funkciu určujúcu jej nasledujúci stav v závislosti na kombinácii stavov buniek v definovanom susedstve. Stavy buniek sú aktualizované synchrónne v disktétnych časových krokoch.

2.2 Formálny popis uniformného celulárneho automatu

Uvažujme automat, ktorého súčasťou je množina buniek, ktoré nadobúdajú určité stavy. Tieto stavy môžeme chápať ako podmnožinu diskrétnych hodnôt Z^D v D-dimenzionálnom priestore. Každá bunka má definované svoje susedstvo. Šablónu susedstva, ktorá je použiteľná pre každú bunku automatu, potom definujeme ako $N = z_0, z_1, ..., z_{n-1}$, kde pre z_i platí, že $z_i \in Z^D$, kde $i \in 0, 1, ..., n-1$. Stav konkrétnej bunky v danom čase t potom označme $s_t(z) \in Q$. Ďalej si definujme lokálnu prechodovú funkciu ako $\Delta : Q^n \rightarrow Q$. Teraz môžeme vývoj tohto automatu v jednotlivých krokoch času zapísať ako:

$$s_{t+1}(z) = \Delta(s_t(z_0 + z), s_t(z_1 + z), ..., s_t(z_{n-1} + z))$$

Z formálnej definície je teda zrejmé, že nový stav danej bunky závisí iba na stave vyšetrovanej bunky a stave jej okolia [2].



Obrázok 2.1: Binárny 1D CA, $r = 1$, pravidlo 250

2.3 Príklad binárneho uniformného 1D CA

- Majme N buniek, každá môže byť v stave 0 alebo 1 (0 = biela bunka, 1 = čierna bunka).
- Lokálna prechodová funkcia je rovnaká pre všetky bunky (uniformný CA).
- Rádius r udáva počet buniek bezprostredne susediacich s bunkou i na každej strane.
- Neexistujúce susedné bunky na okraji celulárnej štruktúry sú považované za log. 0 - konštantné (nulové) okrajové podmienky.
- Stav *i-tej* bunky v čase *t+1* je vypočítaný pomocou lokálnej prechodovej funkcie F.

  Pre $r = 3$:

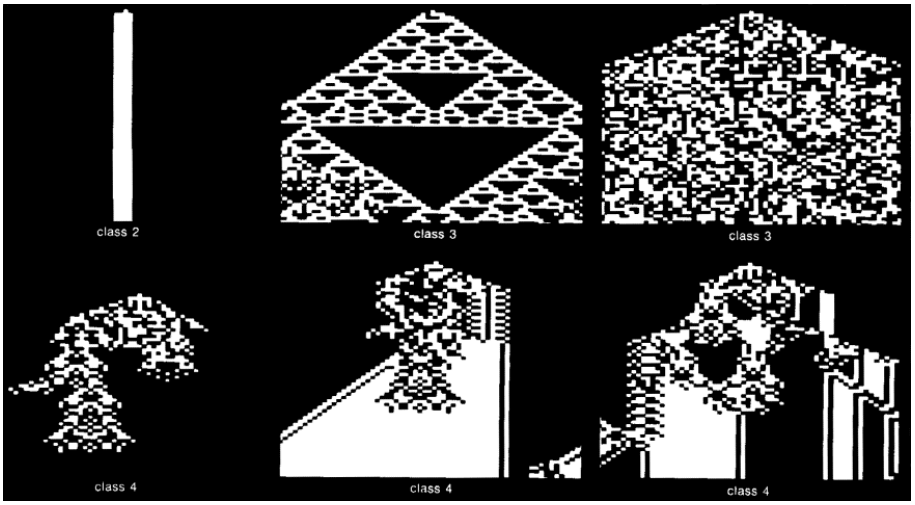
  $Q_{t+1}^{i} = F(Q_t^{i-3}, Q_t^{i-2}, Q_t^{i-1}, Q_t^{i}, Q_t^{i+1}, Q_t^{i+2}, Q_t^{i+3})$

Pre 1D celulárny automat s rádiusom $r = 1$, pravidlom 250, toto pravidlo nastavuje konkrétnu bunku na 1 (čiernu), pokiaľ aspoň jedna bunka z jej bezprostredných susedov z predchádzajúceho stavu celulárneho automatu je čierna. A naopak, ak sú obe susedné bunky biele, aj aktuálne počítaná bunka je nastavená na 0. S počiatočným stavom, kde je jediná bunka čierna, toto pravidlo smeruje k šachovnicovému vzoru [8]. Priebeh tohto pravidla môžme vidieť na obrázku 2.1.

2.4 Kategórie celulárnych automatov

Každé z pravidiel vedie k vzorom, ktoré sa rozlišujú v detailoch. Všetky tieto vzory však spadajú do 4 tried [10]:

- **trieda 1** - vývoj vedie k homogénnemu štádiu, v ktorom, napríklad, všetky bunky majú hodnotu 0,



Obrázok 2.2: Vývoj niektorých typických celulárnych automatov z náhodných počiatočných stavov

- **trieda 2** - vývoj vedie k nehomogénnej množine stabilných alebo periodických štruktúr, ktoré sú oddelené a jednoduché,
- **trieda 3** - vývoj vedie k chaotickým vzorom,
- **trieda 4** - vývoj vedie ku komplexným štruktúram, niekedy dlhotrvácnym.

Príklady týchto tried môžeme vidieť na obrázku 2.2.

Existencia iba štyroch kvalitatívnych tried znamená značnú univerzálnosť v správaní celulárnych automatov. Veľa výhod celulárnych automatov závisí iba na konkrétnej triede, v ktorej sa nachádzajú a nie na presných detailoch ich evolúcie.

K analýze tried celulárnych automatov nám poslúži napríklad stupeň predvídateľnosti výstupu evolúcie celulárneho automatu s vedomosťou o počiatočnom stave.

Pre triedu 1 je kompletná predikcia jednoduchá: nezáležiac na počiatočnom stave, evolúcia systému vždy dôjde k unikátnemu homogénnemu stavu.

Trieda 2 celulárnych automatov má vlastnosť, ktorá propaguje efekty konkrétnej hodnoty iba do konečnej vzdialenosti, to znamená iba do konečného množstva susediacich hodnôt. Zmena jedinej hodnoty počiatočného stavu ovplyvní konečný región hodnôt okolo, dokonca aj po nekonečnom množstve časových krokov. Toto správanie ukazuje, že predvídateľnosť konkrétneho konečného stavu hodnoty vyžaduje vedomosť iba konečnej množiny počiatočných hodnôt.

V porovnaní s celulárnymi automatmi triedy 2, v triede 3 zmena počiatočnej hodnoty je takmer stále donekonečna propagovaná konečnou rýchlosťou a teda ovplyvňuje hodnoty stále vzdialenejšie s narastajúcim časom. Aby sme v celulárnych automatoch triedy 3 vedeli predvídať hodnotu bunky v nekonečnom čase, museli by sme vedieť nekonečné množstvo počiatočných konfigurácií.

Celulárne automaty triedy 4 sa od predchádzajúcej triedy líšia ešte väčším stupňom nepredvídateľnosti [11].

Kapitola 3

Mikroskopická simulácia dopravy

Počítačové modely sú často používané pri analýzach systémov premávky. Modely simulácie premávky môžu byť rozdelené na základe požadovaného detailu, akým je tok premávky modelovaný a spôsobu, ktorým je premávka v sieti riadená. Najdetailnejšie modelujú individuálne vozidlá a sú bežne používané na vytváranie riadiacich plánov lokálnej dopravy. S takýmto typom simulácie dokážeme modelovať komplexné prípojné pruhy na diaľniciach a siete so zápchami a zároveň poskytnúť vizuálnu reprezentáciu efektov na operáciách premávky.

Mikrosimulácia môže byť použitá na vývoj nových systémov a optimalizáciu ich efektivity. Jednoducho môžu odhadnúť dopad nových schém vytváraním výstupov na širokej škále meraní efektivity. Mnohé z týchto dopadov, ako napríklad množstvo vypúšťaných emisií (obrázok 3.1), je často obtiažne merať v reálnom prostredí [26].

3.1 Mikrosimulácia

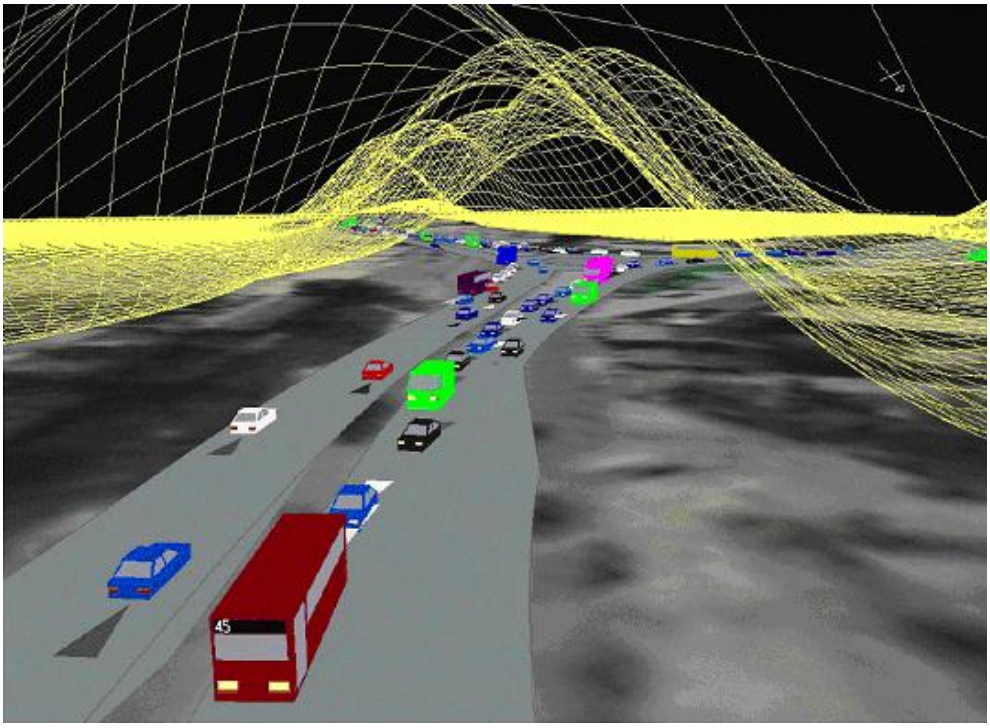
Mikrosimulačné modely sú počítačové modely pracujúce na úrovni správania individuálnych entít, ako sú ľudia, rodiny, alebo autá [29]. Takéto modely simulujú veľké populácie skladajúce sa z takýchto nízkoúrovňových entít, aby ukázali globálny dopad na celkový systém ich jednotlivým správaním.

3.1.1 Makrosimulácia

Na rozdiel od mikrosimulácií, v makroskopických prístupoch nie je v popredí dynamika individuálnych vozidiel, ale dynamika kvantít, ktoré majú makroskopický význam. Vo všeobecnosti sa do úvahy berú hustota áut $\rho(x,t)$ a priemerná rýchlosť $v(x,t)$, ktoré sú obe funkciami priestoru x a času t. Riešenie makroskopického prístupu môže byť realizované analyticky alebo pomocou simulácie. Pokiaľ sa jedná o ohodnotenie jedného segmentu cesty, analytické riešenia sa ešte dajú použiť, ale keď už berieme do úvahy časové a priestorové interakcie dopravných tokov v cestnej sieti, používané metódy bývajú obvykle simulačné.

3.2 Jednoduchý model

Prvé štúdie používania celulárnych automatov pre simuláciu dopravy boli predstavené Nagelom a Schreckenbergom [5], ktorí vytvorili jednoduchý stochastický CA model pre simuláciu



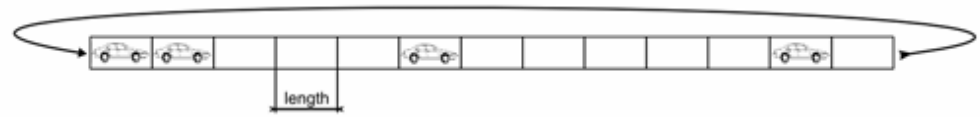
Obrázok 3.1: Miera vypúšťaných emisií premávkou v cestnej sieti [26]

jednopruhovej diaľničnej premávky s otvorenými alebo cyklickými hraničnými podmienkami. Každá bunka celulárneho automatu predstavovala segment cesty 3.2. Lokálna prechodová funkcia (pravidlo) definuje nový stav bunky na základe vlastného súčasného stavu a stavu jej susediacich buniek. Jedná sa teda o diskrétne pohyby vozidiel z jednej bunky do nasledujúcej.

Priestorovo predstavuje každá bunka iba vopred definovanú konštantnú dĺžku segmentu cesty. Priestor je teda hrubozrnný, čo odlišuje tento model od klasických mikroskopických modelov, ktoré bežne používajú polospojitý priestor. Najviac sa používa 7,5m dĺžka bunky, pretože pri zápche zaberá každé auto približne takúto veľkosť [5]. Táto vzdialenosť zahŕňa potrebnú medzeru pred alebo za autom. Aby bolo v CA modeloch možné simulovať aj autá, ktorých dĺžka presahuje túto hodnotu, je nutné buď používať bunky, ktoré reprezentujú dĺžku s inou hodnotou, alebo umiestňovať jedno vozidlo do viacerých buniek súčasne [3]. Druhá možnosť obvykle poskytuje výhodnejšie riešenie napríklad z hľadiska budúceho rozširovania modelu o nové, dlhšie vozidlá. Ďalšou výhodou použitia takéhoto prístupu je jeho presnosť. Čím máme pomocou prvého prístupu bunky väčšie, tým nám klesá presnosť modelu a výsledky sú viac a viac skresľované [3].

3.2.1 Lokálna prechodová funkcia

Vlastnosti premávky s jedným pruhom bývajú modelované na základe integer-ovej hodnoty pravdepodobnosti pravidiel celulárnych automatov [3]. Pre ľubovoľnú konfiguráciu, jeden časový skok systému pozostáva z nasledujúcich štyroch krokov, ktoré sa pre všetky autá vykonávajú paralelne:



Obrázok 3.2: Dopravná sieť postavená na celulárnych automatoch s cyklickými hraničnými podmienkami [5]

- **Akcelerácia** - ak je aktuálna rýchlosť v_v menšia ako maximálna rýchlosť v_{max} ($v_v < v_{max}$) a zároveň medzera pred bunkou i $\gamma(i)$ je väčšia ako aktuálna rýchlosť ($\gamma(i) > v_v$), potom $v_v = v_v + 1$.
- **Spomaľovanie** - ak vozidlo na mieste i vidí nasledujúce vozidlo na mieste $i + j$ ($j \leq v_v$), zníži svoju rýchlosť na $v_v = j - 1$.
- **Náhodnosť** - s pravdepodobnosťou p sa rýchlosť každého vozidla (ak je $v_v > 0$) zníži na $v_v = v_v - 1$.
- **Pohyb vozidiel** - každé vozidlo je posunuté o v_v miest.

Takýto jednoduchý model dopravy postavený na celulárnych automatoch dáva netriviálne a realistické správanie [5]. Tretí krok je v simulácii dopravy veľmi dôležitý, lebo bez neho by bola dynamika úplne deterministická a s ľubovoľnou počiatočnou konfiguráciou by vždy dosiahla stacionárny vzor.

3.3 Iné modely dopravy

Medzi ďalšie prístupy by sme mohli zaradiť napríklad model Benjamina, Johnsona a Huia [1]. Ide o veľmi podobný model, ako Nagel-Schreckenberg [5], ale s pridaním pravidla označeným „slow-to-start“. To znamená, že vozidlo, ktoré zastaví na nulovú rýchlosť, sa opäť pohne pri prvej príležitosti s pravdepodobnosťou $1 - p_{slow}$ a v nasledujúcom časovom okamžiku s pravdepodobnosťou p_{slow}. Autori použili tento model k štúdiu efektu prípojných pruhov na diaľniciach so zistením, že znížením maximálnej povolenej rýchlosti v pruhu najbližšom k tomu prípojnému sa badateľne zníži dĺžka radu áut čakajúceho na vstup na diaľnicu.

3.4 Spôsoby urýchľovania simulačných nástrojov dopravy

Pri simulátoroch dopravy väčšinou očakávame, že budeme mať výsledky ich behu čo najskôr. To znamená, že by sme potrebovali analyzovať výstup simulácie tak, aby sme boli schopní vhodne a včas zareagovať na to, čo nám takáto simulácia prináša. Toto vieme dosiahnuť rôznymi optimalizáciami a heuristikami. Medzi najdôležitejšie môžeme zaradiť paralelné výpočty, ktoré sú už v dnešnej dobe dostupné takmer na každom stolnom, ale aj prenosnom počítači.

Lee a Chandrasekar [4] popísali a demonštrovali metódu behu paralelnej simulácie. Metóda spočíva v tom, že spúšťa niekoľko inštancií programu naraz paralelne. Podstatou bolo rozdeliť simulovanú sieť na niekoľko regiónov a simulovať rôzne inštancie programu simultánne tak, aby bol možný presun áut na krajoch regiónov. Metóda je demonštrovaná s použitím simulačného nástroja Paramics [49] a jeho programovateľného rozhrania

na multiprocesorovom systéme UNIX. Rýchlosť simulácie je ovplyvňovaná množstvom vozidiel simulovaných v konkrétnej inštancii. Výsledky z použitia takejto metódy ukazujú zrýchlenie času simulácie 1,5 až 2,25-násobne pri dvoch procesoroch a od 1,75 po 3,75-násobné zrýchlenie s tromi procesormi v porovnaní s rýchlosťou simulácie bez paralelného behu.

V článku Strippgena a Nagela [9] je popísaná metóda multiagentnej simulácie dopravy za použitia architektúry CUDA. Ide o architektúru súčasných grafických kariet (GPU) s množstvom výpočtových elementov (multiprocesorov). Táto architektúra je popísaná v kapitole 5.3. Implementáciou simulátora dopravy pomocou tejto architektúry autori dosiahli 67-násobné zrýchlenie oproti optimalizovanej verzii napísanej v jazyku Java.

Existujú práce, ktoré sa zaoberajú urýchľovaním simulácie dopravy pomocou rekonfigurovateľných polí FPGA, ako je to napríklad v článku [28]. Autori demonštrovali použitie 64-bitových mikroprocesorov a FPGA vo vysokorýchlostnom prepojení s nízkou odozvou na veľkej metropolitnej oblasti. Ďalej sa v tejto práci tvrdí, že predchádzajúce simulátory dopravy využívajúce FPGA boli limitované malými segmentami ciest, alebo muselo byť použité veľké množstvo FPGA zariadení. S využitím jedného zariadenia FPGA dosiahli 12.8 násobné zrýchlenie oproti mikroprocesoru AMD.

Kapitola 4

Vývoj architektúr NVIDIA GPU

4.1 NVIDIA Quadro

Dizajnéri série kariet Quadro pre rozhrania AGP, PCI a PCI Express sa zameriavajú na akceleráciu CAD (Computer-Aided Design) a DCC (Digital Content Creation) [40] a karty sú väčšinou inštalované v pracovných staniciach (v porovnaní s produktovou líniou GeForce, ktorých primárnym cieľom sú počítačové hry). Medzi konkurenčné karty patrí línia FireGL grafických kariet pre pracovné stanice od spoločnosti ATI.

4.2 NVIDIA Tesla

Línia Tesla je tretia značka grafických kariet NVIDIA. Je postavená na high-end grafických kartách od čipu G80, takisto ako karty Quadro. Tesla karty sú prvé karty spoločnosti NVIDIA vyhradené pre všeobecné výpočty na grafických kartách (GPGPU viď. 5.1).

4.3 Grafické karty pred radou GeForce

Prvá karta firmy NVIDIA s označením NV1 bola predstavená v roku 1995. Táto mala rozhranie PCI, grafickú pamäť 2 alebo 4 MB [33]. Ďalej nasledovali karty s označením RIVA a Vanta, ktoré už všetky boli dostupné aj v prevedení pre rozhranie AGP. Najvýkonnejšia z týchto kariet bola RIVA TNT2 Ultra, ktorá prišla na trh v roku 1999. RIVA TNT2 Ultra podporovala API DirectX 6 (kapitola 4.8) a OpenGL 1.1 a pamäťovú priepustnosť mala 2,9 GB/s [41].

4.4 GeForce 256 - GeForce 7 (7xxx)

4.4.1 GeForce 256

Tieto karty boli od roku 1999 vyrábané iba pre rozhranie AGP 4x v dvoch prevedeniach: buď s pamäťami SDR [1] alebo s pamäťami DDR [2] o veľkosti 32 alebo 64MB. Rozdiel medzi nimi bol badateľný hlavne v priepustnosti pamätí, a to 2,7 GB/s s SDR a 4,8 GB/s s DDR pamäťami [27].

[1] SDRAM - Synchronous Dynamic Random Access Memory
[2] DDR SDRAM - Double Data Rate SDRAM

Ako prvé prišli s hardware-ovou podporou pre transformácie a osvetľovanie (T&L), ale kritici jej neprikladali veľkú váhu. Spočiatku bola použiteľná iba v niektorých OpenGL 3D first-person tituloch, najznámejším bol Quake III Arena. 3dfx a ďalší konkurenční výrobcovia grafických kariet tvrdili, že rýchle CPU nahradí chýbajúcu jednotku T&L. To sa však neskôr ukázalo ako nie veľmi rozumné, hlavne kvôli tomu, že poprední výrobcovia grafických benchmark testov (Futuremark 3DMark 2000) využívali T&L vo veľkom, čo zanechávalo karty bez T&L hardware-ovej podpory na chvoste za aj nízkorozpočtovými kartami ako napr. GeForce 2 MX.

4.4.2 GeForce 2 a GeForce 3

GeForce 3 je tretia generácia grafických kariet GeForce a bola predstavená v roku 2001. Oproti predošlým modelom GeForce 3 pridáva programovateľný pixel a vertex shader, plnoscénový multi-samplovací anti-aliasing a vylepšený všeobecný proces renderovania. Odnož GeForce 3 známa ako NV2A bola používaná v konzolách Xbox firmy Microsoft.

Karta GeForce 3 bola prvá na trhu podporujúca Direct3D 8. Jej programovateľný shader dovoľoval aplikáciám vytvárať vlastné vizuálne programy s efektami v jazyku Microsoft Shader 1.1.

Pre lepšie využívanie dostupného výkonu pamäti mala GeForce 3 hardware-ový manažér pamäti *Lightspeed Memory Architecture* (LMA). Toto pozostáva z niekoľkých mechanizmov, ktoré redukujú prekresľovanie, znižujú prenosy pamäti komprimovaním z-buffera (hĺbkového buffera) a lepšie manažujú pamäťovú zbernicu [24].

4.4.3 GeForce 4

Karty GeForce 4 boli veľmi podobné svojim predchodcom GeForce 3. Najväčšími zmenami, ktorými prešli tieto karty, boli väčšia frekvencia jadra a pamäti, vylepšený kontrolér pamäti (Lightspeed Memory Architecture II), ďalší vertex shader, hardware-ový anti-aliasing a podpora prehrávania DVD [23].

4.4.4 GeForce FX (5xxx)

S neustálym vývojom real-time 3D grafickej technológie sa objavuje DirectX 9.0 vylepšujúce technológiu programovateľného pipeline s príchodom Shader Model 2.0. Séria GeForce FX je prvá generácia kariet NVIDIA kompatibilná s DirectX 9.

Séria bola vyrábaná so 130nm technológiou [38] a s 256 bitovou zbernicou [36] a ich čipy pozostávali zo 125 miliónov tranzistorov. Hlavným dôvodom zvýšenia počtu tranzistorov je fakt, že GPU GeForce FX plne podporuje plávajúcu desatinnú čiarku. Prináša rad nových pamäťových technológií, medzi ktoré patria DDR2, GDDR-2 a GDDR-3 [3] [22].

Séria priniesla vylepšenia hardware-ového spracovávania videa (Video Processing Engine - VPE).

Výkon kariet s DirectX 7 a DirectX 8 je výborný v porovnaní s konkurenciou, ale keď sa začne hovoriť o DirectX 9, sú oveľa menej konkurencieschopné. Tieto slabosti sú spôsobené tým, že čipy NV3x sú navrhnuté s menšou priepustnosťou paralelizmu a výpočtov ako konkurenčné výrobky [15]. Je obtiažne dosiahnuť vysokú efektivitu s architektúrou kvôli slabinám architektúry a veľkej závislosťou od dobre optimalizovaného kódu pixel shared-u [15].

[3] Graphics Double Data Rate 3

4.4.5 GeForce 6 (6xxx) a GeForce 7 (7xxx)

Séria GeForce 6 prináša niekoľko nových a vylepšených technológií od PureVideo, následného spracovania (post-processing) videa, technológie SLI a podporu pre Shader Model 3.0 (vyhovujúci špecifikácii Microsoft DirectX 9.0c a OpenGL 2.0).

SLI

Scalable Link Interface je názov pre multi-GPU riešenie spojením 2 alebo viac grafických kariet súčasne tak, aby vo výsledku formovali jediný výstup. Táto technológia bola vyvinutá ako aplikácia paralelného spracovania pre počítačovú grafiku za účelom zvýšenia potenciálu spracovania pre grafiku [42].

Technológiu SLI prvýkrát použila firma 3dfx pod názvom Scan-Line Interleave v roku 1998. Táto technológia bola používaná v kartách Voodoo2. Po odkúpení 3dfx firma NVIDIA získala túto technológiu, ale nepoužívala ju. NVIDIA neskôr predstavila SLI v roku 2004 s predstavou použitia v moderných počítačových systémoch postavených na zbernici PCI Express [4].

PureVideo

PureVideo je hardware-ová technológia vytvorená pre prenesenie potreby dekódovania videa a post-processing-u videa z CPU na grafické karty NVIDIA od série GeForce 6, GeForce M (mobilné karty) a Quadro. PureVideo pracuje priamo so software-om prehrávania médií.

V kartách GeForce 6 bolo PureVideo rozšírené o dekódovanie pokročilejších kodekov (MPEG-4, WMV9), oproti pôvodnému MPEG-2, vylepšený post-processing (odstránenie prekladania obrazu) a pod [13].

IntelliSample 4.0

IntelliSample je značka firmy NVIDIA pre metódu anti-aliasing na GeForce grafických kartách. Verzia 4.0 sa používa v sériách GeForce 6 a GeForce 7 a používa 2 nové metódy oproti predošlej verzii: Transparentný Supersampling (TSAA) a rýchlejši, ale s nižšou kvalitou Transparentný Multisampling (TMAA). Tieto metódy boli vyvinuté pre lepšie zobrazovanie antialiasingu scén s čiastočne transparentnými textúrami (ako napríklad reťazový plot) a šikmo naklonenými textúrami k pozorovateľovi [25].

Rad GeForce 6800

Rad 6800 predstavovala najvyšší rad kariet NVIDIA. 16 pixel pipeline-ový model GeForce 6800 Ultra (NV40) bol 2 až 2.5 násobne rýchlejší, ako jeho najsilnejší predchodca rady FX (GeForce FX 5950 Ultra) so štvornásobne väčším množstvom pixel pipeline-ov, dvakrát väčším množstvom textúrovacích jednotiek a s pridanou vylepšenou pixel-shader architektúrou [34].

Prvé benchmark testy ukazujú miernu nevýhodu v porovnaní s kartami firmy ATI v podobnej cenovej hladine [43]. Nové ovládače však zvyšujú výkon a efektivitu produktov oboch firiem. Pri porovnávaní týchto kariet v rôznych aplikáciách tieto karty ukazujú rôzne výsledky, niekedy horšie, inokedy lepšie, v závislosti od danej testovacej aplikácie. Silná stránka

[4] Peripheral Component Interconnect Express (PCIe) - je štandard počítačového rozširujúceho slotu nahradzujúci staršie PCI, PCI-X a AGP štandardy

karty spoločnosti NVIDIA sa ukazuje v aplikáciách naprogramovaných pre OpenGL, kým ATI vedie v mnohých Direct3D aplikáciách.

Do roku 2004 boli všetky karty firmy NVIDIA vyrábané pre zbernicu AGP [5]. NVIDIA pridala neskôr podporu pre zbernicu PCI Express v kartách GeForce 6 obvykle za použitia čipu - most AGP-PCIe. Neskôr, keď grafické karty NVIDIA boli vyrábané pre PCIe natívne, čip obojsmerného mosta im povolil vyrábať ich aj ako AGP karty.

Séria 6800 bola používaná aj v prenosných počítačoch pod označením GeForce Go 6800 a Go 6800 Ultra.

4.5 GeForce 8 (8xxx, G80) a GeForce 9 (9xxx)

Ide o tretiu hlavnú GPU architektúru vyvinutú spoločnosťou NVIDIA a predstavuje prvú architektúru so zjednoteným shader modelom. V porovnaní s predchádzajúcimi sériami kariet má rad GeForce 8 univerzálne procesory (stream procesory) s pohyblivou rádovou čiarkou [35].

Na rozdiel od vektorového spracovania pri starších jednotkách shader-ov každý stream procesor je skalárny a teda dokáže pracovať s iba jedným komponentom súčasne. Toto ich robí jednoduchšími na výrobu s tým, že sú stále pomerne flexibilné a univerzálne. Skalárne shader-e majú taktiež výhodu väčšej efektivity v mnohých prípadoch v porovnaní s predchádzajúcou generáciou vektorových shader-ov, ktoré závisia na konkrétnych inštrukčných postupnostiach a usporiadaní, aby dosahovali vysokú priepustnosť. Nižšia priepustnosť skalárnych procesorov je kompenzovaná efektivitou a behom na vysokých frekvenciách (umožnené ich jednoduchosťou). Rôzne časti jadra kariet GeForce 8 pracujú na rôznych frekvenciách, podobne ako predchádzajúca séria GeForce 7. Napríklad, stream procesory karty GeForce 8800 GTX pracujú na frekvencii 1.35 GHz, zatiaľ čo zvyšok čipu na frekvencii 575 MHz [21].

4.5.1 8800 GTX / 8800 Ultra

Karta 8800 GTX je vybavená 768 MB GDDR3 pamäťami RAM a jej čip sa skladá z 681 miliónov tranzistorov pokrývajúcich plochu 480 mm^2 s technológiou 90nm. GTX má 128 stream procesorov s frekvenciou 1.35 GHz a pamäťovú priepustnosť 86.4 GB/s.

8800 Ultra je identická s architektúrou karty GTX, ale predstavuje vyššie frekvencie shader-ov, jadra a pamätí. Jadro bolo taktované na frekvencii 612 MHz, shader-e na frekvencii 1.5 GHz a pamäte na frekvencii 1080 MHz s teoretickou pamäťovou priepustnosťou 103.7 GB/s.

Obe karty majú 2 SLI porty, ktoré im umožňujú pripojiť naraz 3 NVIDIA karty (3-way SLI). Karty disponujú výpočtovou schopnosťou 1.0 (viď kapitola 5.4.2).

4.5.2 9800 GTX

Zhruba ide o kartu 8800 GTS 512MB s dvomi SLI konektormi, vyššími frekvenciami a podporou pre technológiu Nvidia Hybrid Power. Táto technológia umožňuje GPU vypnúť sa počas behu aplikácií, ktoré nespotrebúvajú veľké množstvo zdrojov a použiť namiesto nej integrovanú GPU kartu. Takisto, ako jej predchodca 8800 GTX, má aj 9800 GTX 128 stream procesorov, líšia sa však vo vedľajšom revíznom čísle výpočtovej schopnosti - 1.1 (5.4.2).

[5] Accelerated Graphics Port - vysokorýchlostný dedikovaný port pre použitie výlučne s grafickými kartami pripojenými do základných dosiek počítača

4.6 GeForce 200 a GeForce 300

Séria GeForce 200 rozširuje výkon a funkčnosť architektúry G80 a prináša (s výnimkou niektorých modelov) dvojitú presnosť čísel s pohyblivou rádovou čiarkou a výpočtovú schopnosť 1.3. Karty GTX 280 a GTX 260 sú postavené na rovnakom jadre. Počas výroby kariet GTX sú tieto karty testované voči defektom logickej funkčnosti jadra. Tie, ktoré nevyhovujú hardware-ovej špecifikácii GTX 280, sú znovu testované a nakoniec označené ako GTX 260 (ktorá má v špecifikácii menej stream procesorov, menšiu zbernicu a pod.).

Karty podporujú Direct3D 10.0, prípadne 10.1 (v závislosti od modelu), Shader Model 4.0, prípadne 4.1 a OpenGL 3.3. Každý stream multiprocesor obsahuje 8 stream procesorov a 2 špeciálne funkčné jednotky (SFU). Každý stream procesor môže vykonať 2 operácie MAD [6] s jednoduchou presnosťou [37].

4.7 Architektúra Fermi

Architektúra Fermi je najväčším pokrokom vo svete GPU architektúr od jadier G80 (viď 4.5).

Nová generácia stream multiprocesorov:

- obsahuje 32 CUDA jadier (4x viac oproti architektúre GT200),
- prináša 8-násobný nárast výkonu operácií s dvojitou presnosťou oproti GT200,
- prináša dvojitý plánovač Warp, ktorý simultánne plánuje inštrukcie dvoch nezávislých warp-ov (viď 5.4.1),
- prináša 64 kB RAM s konfigurovateľným rozdelením zdieľanej pamäti a L1 cache [44].

Prvá karta Fermi s 3 miliardami tranzistorov predstavuje 512 CUDA jadier. CUDA jadro vykoná inštrukciu s pohyblivou rádovou čiarkou alebo celým číslom v jednom takte jedného vlákna. 512 CUDA jadier je usporiadaných v 16-tich multiprocesoroch po 32 jadier. GPU má šesť 64-bitových pamäťových partícií pre 384-bitové pamäťové rozhranie, podporujúce pamäť až o veľkosti 6 GB GDDR5 DRAM [44].

4.7.1 Vysokovýkonné CUDA jadrá

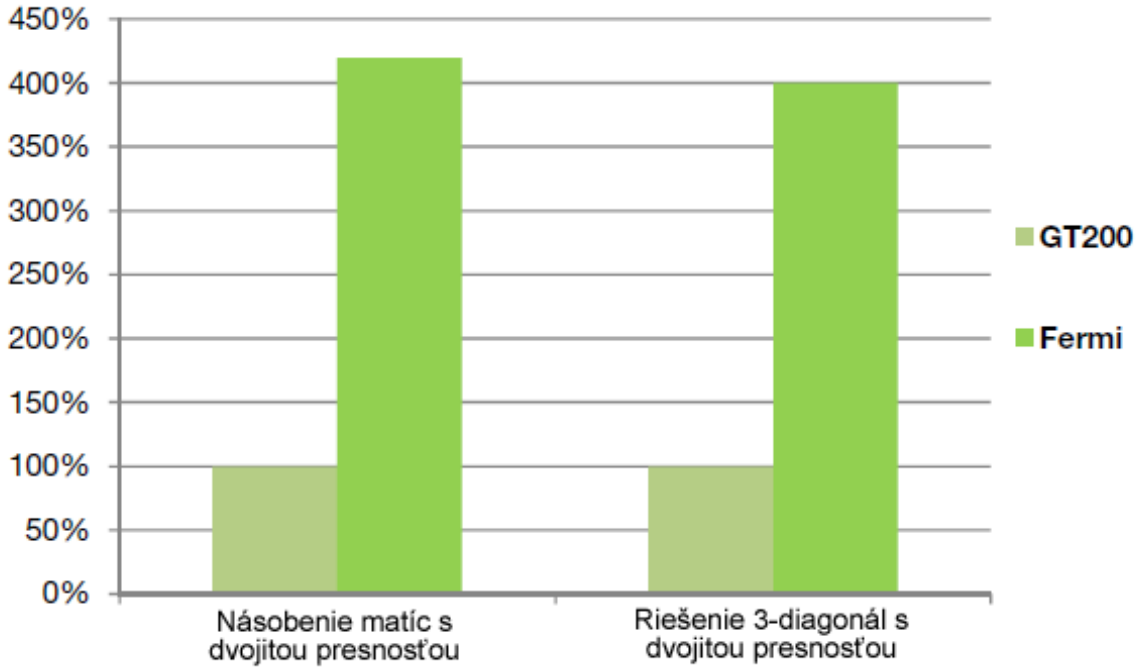
Každý stream multiprocesor predstavuje 32 CUDA procesorov. Každý CUDA procesor má plne pipelinovú logickú jednotku celých čísel (ALU) a jednotku desatinných čísel (FPU). Fermi architektúra používa nový štandard pohyblivej rádovej čiarky IEEE 754-2008, poskytujúci inštrukciu FMA (fused multiply-add) pre jednoduchú aj dvojitú presnosť. FMA je od inštrukcie MAD (multiply-add) vylepšená tým, že vykonáva násobenie a sčítanie v jedinom konečnom zaokrúhľovacom kroku, bez straty presnosti pri sčítaní. Inštrukcia FMA je presnejšia než separátne vykonávané operácie [44].

4.7.2 Dvojitá presnosť

Aritmetika dvojitej presnosti je základom HPC aplikácií [7], ako lineárna algebra, numerické simulácie a kvantová chémia. Fermi architektúra je špeciálne navrhnutá, aby ponúkala vy-

[6] MAD - Multiply-Add, operácia násobenia a sčítania

[7] HPC - High Performance Computing - Vysokovýkonné počítanie - používa superpočítače a počítačové clustre pre riešenie pokročilých výpočtových problémov



Obrázok 4.1: Výkon aplikácií s dvojitou presnosťou na architektúrach Fermi a GT200 [44]

soký výkon v tejto oblasti. Ako vidieť na obrázku 4.1, jedná sa o dramatický nárast výkonu oproti predchádzajúcej architektúre GT200.

4.7.3 Prvé GPU s podporou ECC pamäti

Fermi je prvé GPU s podporou funkcie pre ochranu dát kódom opravy chyby (ECC - Error Correcting Code) v pamäti. ECC je veľmi potrebné v oblastiach, ako je napríklad medicínske zobrazovanie a pri cluster počítačoch. Voľne sa objavujúca radiácia môže spôsobiť zmenu uloženého bitu v pamäti. Technológia ECC detekuje a opravuje jednobitové chyby predtým, ako ovplyvnia systém. Pretože pravdepodobnosť takýchto chýb vzniknutých radiáciou vzrastá lineárne s pribúdajúcim množstvom inštalovaných systémov, ECC je esenciálnou súčasťou vo veľkých cluster-ových inštaláciách [44].

4.7.4 Série GeForce 400 a GeForce 500

Tieto série kariet NVIDIA sú predstaviteľmi architektúry Fermi. Sú vytvárané 40nm technológiou a sú to prvé NVIDIA karty podporujúce OpenGL 4.0 a Direct3D 11. Pri vypustení kariet GeForce 400 na trh nemala ani jedna karta z tohto radu aktívne všetky stream procesory: GTX 480 má jednu skupinu vypnutú, GTX 470 dve skupiny a jeden pamäťový kontrolér vypnutý a GTX 465 má vypnutých 5 skupín procesorov a 2 pamäťové kontroléry. Karty GeForce prichádzajú s 256 MB pripojenými ku každému zapnutému pamäťovému kontroléru v rozmedzí 1 až 1.5 GB. Pre porovnanie, Tesla C2050 má 512 na každej zo šiestich kontrolérov a Tesla C2070 má 1024 MB na každý kontrolér.

Séria GeForce 500 je modifikovaná verzia kariet GeForce 400, hlavne čo sa výkonu a spotreby energie týka. Takisto, ako GeForce 400, aj GeForce 500 podporuje DirectX 11,

GPU	G80	GT200	Fermi
Tranzistorov	681 miliónov	1.4 miliardy	3 miliardy
CUDA jadier	128	240	512
Schopnosť dvojitej presnosti pohyblivej rádovej čiarky	žiadna	30 FMA op / takt	256 FMA op / takt
Schopnosť jednoduchej presnosti pohyblivej rádovej čiarky	128 MAD op / takt	240 MAD op / takt	512 MAD op / takt
Jednotky špeciálnych funkcií SFU / SM	2	2	4
Plánovače Warp / SM	1	1	2
Zdieľaná pamäť / SM	16 kB	16 kB	48 kB alebo 16 kB konfigurovateľné
L1 Cache / SM	žiadna	žiadna	48 kB alebo 16 kB, konfigurovateľné
L2 Cache	žiadna	žiadna	768 kB
ECC pamäť	nie	nie	áno
konkurentné kernely	nie	nie	až 16
Load/Store šírka adresy	32-bitová	32-bitová	64-bitová

Tabuľka 4.1: Porovnanie troch prelomových GPGPU NVIDIA architektúr

OpenGL 4.1 a OpenCL 1.0. Boli vytvorené preto, aby mohli konkurovať kartám série Radeon HD 6000 firmy AMD (ATI). Jedná sa o plne aktívnu radu kariet s architektúrou Fermi, so všetkými 16-timi stream multiprocesormi a všetkými šiestimi 64-bitovými pamäťovými kontrolérmi aktívnymi.

4.8 DirectX

DirectX je kolekcia aplikačného programovateľného rozhrania - API [8] pre zaobchádzanie s multimédiami a ich súvislosťami, hlavne programovanie hier a práca s videom, na platforme Microsoft. Pôvodne názvy týchto API pochádzajú všetky z názvu Direct, ako napríklad Direct3D pre prácu s 3D grafikou, DirectDraw pre prácu s 2D grafikou, DirectSound pre prácu so zvukom, DirectPlay pre komunikáciu po počítačovej sieti atď. Názov DirectX je teda skratka pre všetky API súčasne, písmeno *X* reprezentujúce konkrétne API.

4.8.1 Direct3D

Direct3D je 3D grafické API vrámci DirectX široko používané pri vývoji video hier pre Microsoft Windows, Microsoft Xbox a Microsoft Xbox 360. Direct3D je takisto používané iným software-om pre vizualizáciu a grafické úlohy ako je napríklad CAD/CAM inžinierstvo. Keďže je Direct3D najpoužívanejšie a najpublikovanejšie API z DirectX, je bežné, že sa názvy DirectX a Direct3D vzájomne vymieňajú.

[8] Application Programming Interface

4.9 OpenGL

OpenGL je prostredie pre vytváranie prenosných, interaktívnych 2D a 3D aplikácií. Od jeho vzniku v roku 1992 sa OpenGL stalo široko používaným a podporovaným 2D a 3D programovým API grafických aplikácií, prinášajúcim tisíce aplikácií širokej škále počítačových platforiem. OpenGL pomáha urýchľovať vývoj aplikácií zahrnutím vizualizačných funkcií pre renderovanie, mapovanie textúr, špeciálnych efektov a ďalších [47].

Kapitola 5

Všeobecné výpočty na grafických kartách

5.1 GPGPU

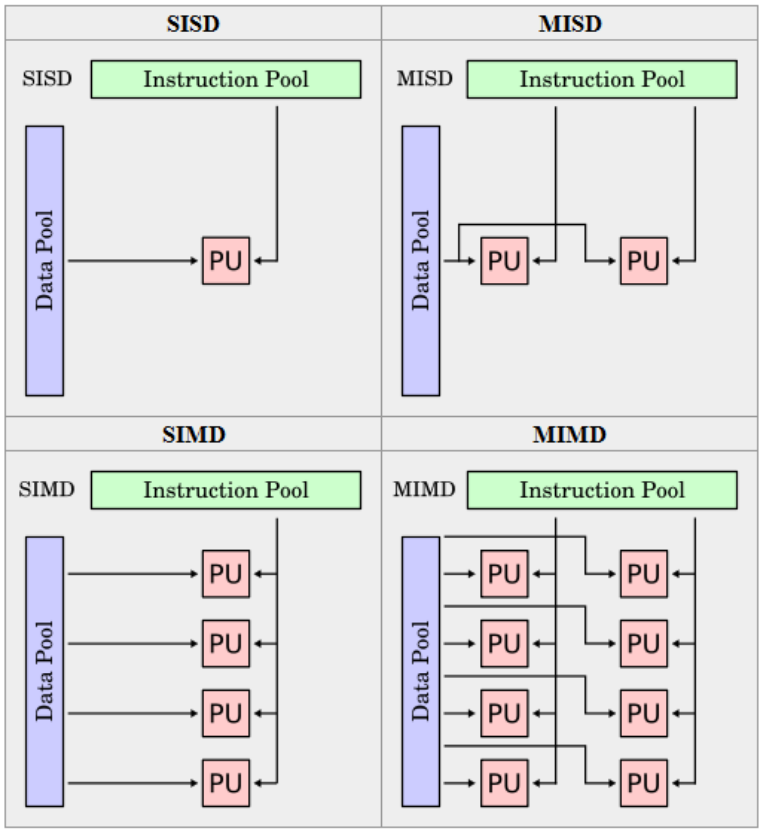
GPGPU je skratka pre *General-Purpose computation on Graphics Processing Units* [1], známe tiež ako *GPU počítanie*. Grafické karty sú vysokovýkonné mnohojadrové procesory s vysokým výpočtovým potenciálom a dátovou priepustnosťou [12]. Grafické karty boli kedysi navrhnuté iba pre účel počítačovej grafiky. Boli obtiažne programovateľné. Dnešné grafické karty sú paralelné procesory využiteľné na všeobecné výpočty s podporou dostupných programovateľných rozhraní a štandardných jazykov, ako napr. jazyk C. Vývojári, ktorí vytvárajú aplikácie na GPU, často dostávajú urýchlenia v násobkoch v porovnaní s optimalizovanou verziou na CPU.

5.2 Flynnová taxonómia

Michael Flynn, emeritný profesor Stanford University, vymyslel metódu klasifikácie počítačových architektúr. Táto klasifikačná metóda je známa ako Flynnová taxonómia [16]. Taxonómia je postavená na počte konkurentných inštrukcií a dátových stream-ov dostupných v architektúre. Rozdelenie je nasledovné 5.1:

- **SISD - Single Instruction, Single Data** [2] - je sekvenčný počítač, ktorý neprináša žiadny paralelizmus, ako na strane inštrukcií, tak ani na strane dátových stream-ov. Príklad SISD architektúry sú tradičné jednoprocesorové stroje.
- **SIMD - Single Instruction, Multiple Data** [3] - je počítač, ktorý spúšťa viacero dátových stream-ov na jednom inštrukčnom stream-e.
- **MISD - Multiple Instruction, Single Data** [4] - v takejto architektúre veľa funkčných jednotiek prevádza rôzne operácie na rovnakých dátach. Patria sem napríklad pipeline-ové architektúry

[1] Užitie grafických kariet pre všeobecné výpočty
[2] Jedna inštrukcia, jedny dáta
[3] Jedna inštrukcia, viacero dát
[4] Viacero inštrukcií, jedny dáta



Obrázok 5.1: Flynnová taxonómia - klasifikácia počítačových architektúr [18]

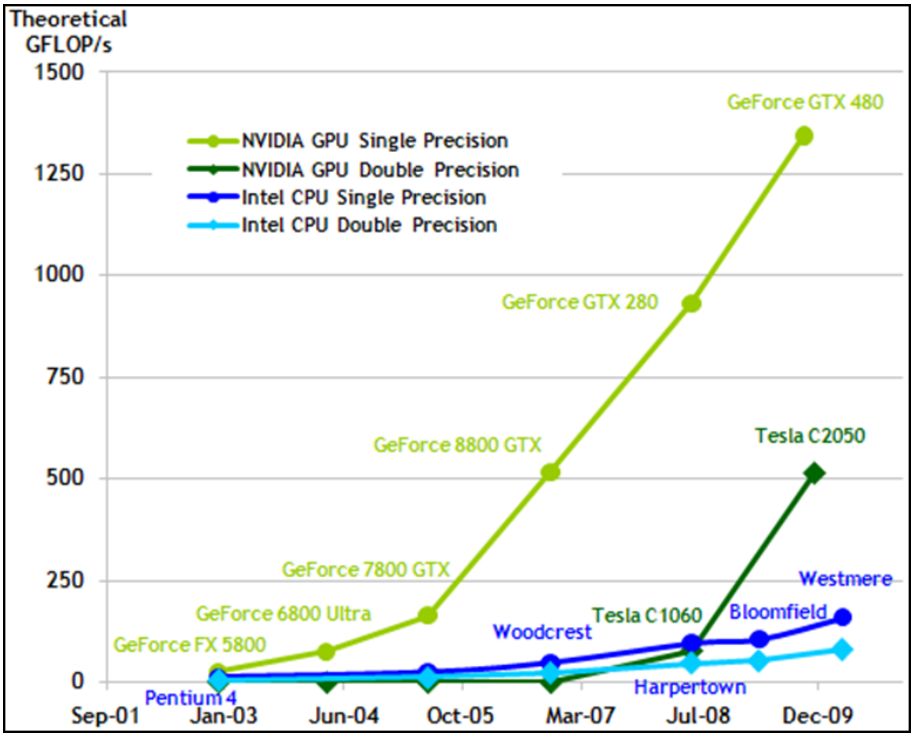
- **MIMD - Multiple Instruction, Multiple Data** [5] - stroje používajúce MIMD majú veľa procesorov, ktoré pracujú asynchrónne a nezávisle

5.2.1 SIMD architektúra

V multiprocesorovom systéme uskutočnením jednej množiny inštrukcií dosiahneme dátový paralelizmus, ak každý procesor vykonáva rovnakú operáciu s rôznymi časťami distribuovaných dát. V niektorých situáciách jedno spúšťané vlákno kontroluje operácie na všetkých častiach dát. V iných rôzne vlákna kontrolujú operáciu, ale spúšťajú rovnaký kód.

Zoberme si napríklad systém s dvomi procesormi (CPU A a B) v paralelnom prostredí a prajeme si vykonať nejakú úlohu na dátach d. Je možné nastaviť CPU A, aby vykonával túto úlohu na jednej časti dát d a CPU B na druhej časti simultánne a teda znížime čas výpočtu. Ako konkrétny príklad si môžme predstaviť sčítanie dvoch matíc. V SIMD implementácii by CPU A mohol sčítať všetky elementy hornej polovice matíc a CPU B spodnej polovice. Keďže oba procesory pracujú paralelne, dosiahli by sme dvojnásobné zrýchlenie výpočtu sčítania matíc oproti situácii, kde by sme mali k dispozícii iba jeden CPU. Teraz si môžme predstaviť, že namiesto dvoch máme 200 procesorov (ako má väčšina moderných GPU). Bolo by teda možné dosiahnuť 200-násobné zrýchelnie pri spustení takejto oprácie na GPU. Toto je však iba teoretický koncept. V skutočnosti musíme počítať aj s réžiou

[5] Viacero inštrukcií, viacero dát



Obrázok 5.2: Operácie s plávajúcou desatinnou čiarkou za sekundu pre CPU a GPU [6]

spojenou s použitím GPU na počítači, ako je napríklad kopírovanie dát do grafickej pamäti a následne po vykonaní operácie výsledné dáta späť z grafickej pamäti, ďalej výmena dát medzi globálnou a zdieľanou pamäťou, atď. Takže 200-násobné zrýchlenie v skutočnosti nedostaneme, ale stále budeme schopní získať badateľné zrýchlenie [16].

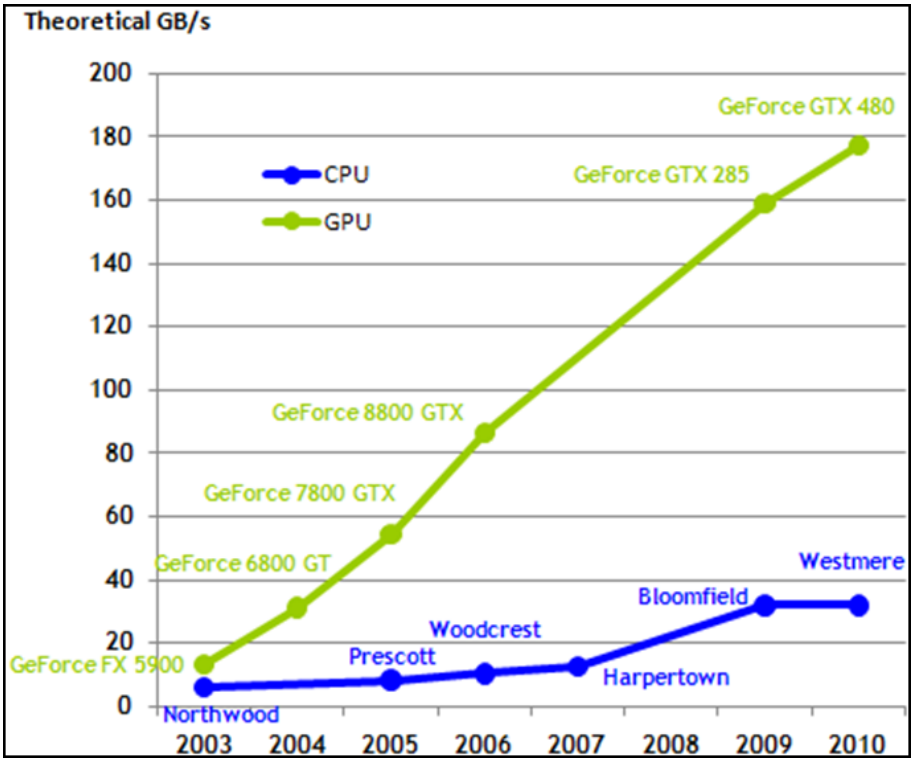
5.3 CUDA

Programovateľný grafický procesor (GPU) sa vyvinul vo vysoko paralelizovateľný, multi-vláknový, mnohojadrový procesor s veľkou výpočtovou silou a obrovskou pamäťovou priepustnosťou, ako je vidieť na obrázkoch 5.2 a 5.3.

5.3.1 Škálovateľný programovací model CUDA

V súčasnosti je výzvou vyvíjať software, ktorý transparentne škáluje svoj paralelizmus tak, aby naplno využil narastajúci počet jadier procesoru.

Základom architektúry CUDA sú 3 kľúčové veci: hierarchia skupín vlákien, zdieľaná pamäť a barierová synchronizácia. Tieto abstrakcie poskytujú jemnozrnné paralelizovanie dát a vlákien vložených do hrubozrnného paralelizmu dát a úloh. Poskytujú programátorovi rozložiť problém na hrubé podproblémy, ktoré môžu byť riešené nezávisle paralelne a potom do jemnejších častí, ktoré zase môžu byť riešené kooperatívne paralelne [6].



Obrázok 5.3: Porovnanie pamäťovej priepustnosti pre CPU a GPU [6]

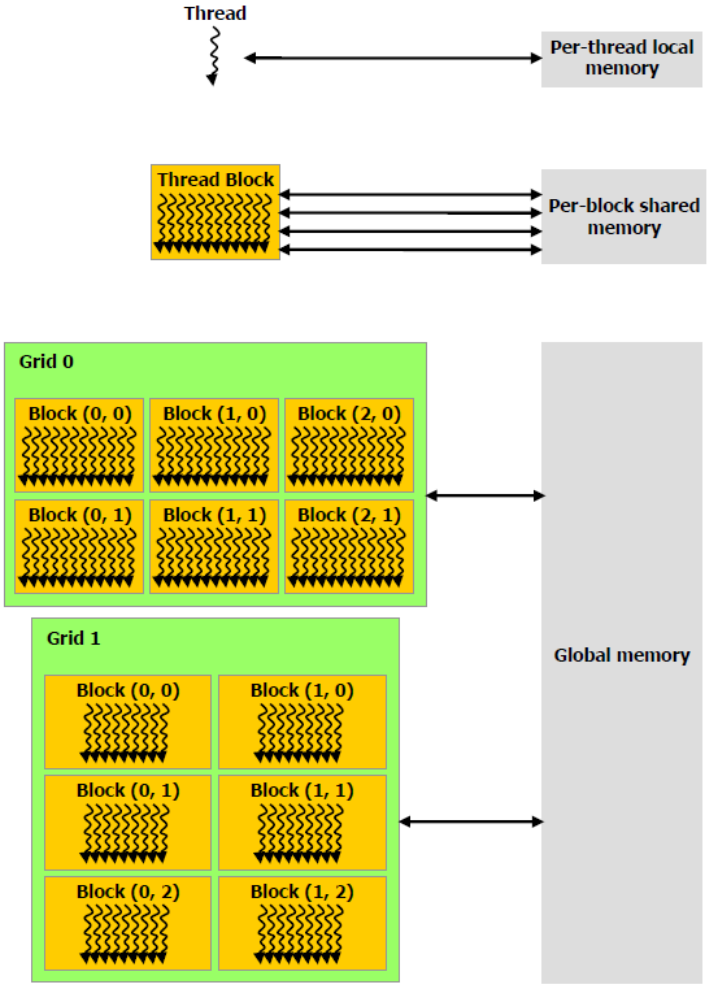
5.3.2 Hierarchia pamäti

Vlákna architektúry CUDA môžu k dátam pristupovať z viacerých pamäťových priestorov počas svojho behu, ako vidieť na obrázku 5.4. Každé vlákno má privátnu lokálnu pamäť. Každý blok vlákien má zdieľanú pamäť viditeľnú všetkým vláknam bloku a s rovnakou životnosťou ako je životnosť bloku. A nakoniec všetky vlákna (všetkých blokov v mriežke) majú prístup k jednej globálnej pamäti [6].

5.3.3 Host a Device

Programovací model architektúry CUDA predpokladá, že CUDA vlákna sú spúšťané na fyzicky odlišnom zariadení (device), ktoré sa správa ako koprocesor k hosťovskému zariadeniu (host), na ktorom je spustený program C. Toto je prípad, keď sú napríklad kernely spúšťané na GPU a zvyšok programu C je vykonávaný CPU [6].

CUDA programovací model takisto predpokladá, že obe zariadenia, aj host, aj device, si spravujú vlastnú DRAM, ako pamäť device a pamäť host. Z toho vyplýva, že program riadi globálnu, konštantnú a textúrovaciu pamäť viditeľnú kernelom cez volania do CUDA runtime. Toto zahŕňa alokáciu a dealokáciu pamäti device, takisto ako prenosy dát medzi pamäťou host a device [6].



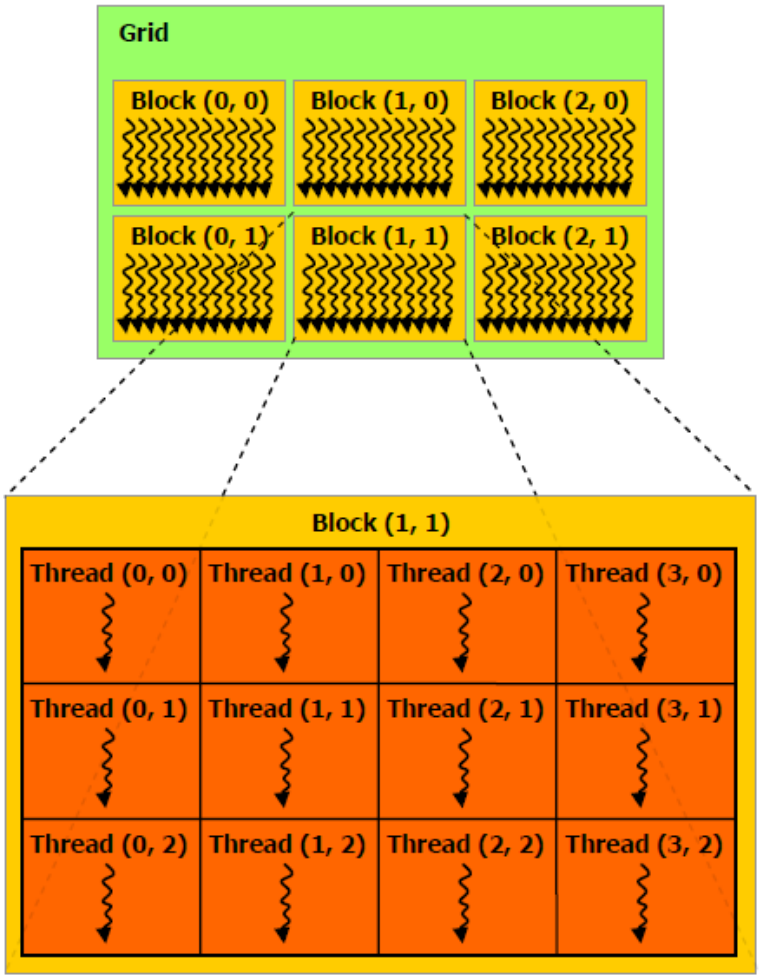
Obrázok 5.4: Prístup k pamäti architektúry CUDA [6]

5.4 OpenCL

OpenCL (Open Computing Language) je prvý otvorený, neplatený štandard pre paralelné programovanie, ktoré je vykonávané na heterogénnych platformách skladajúcich sa z CPU, GPU a iných procesorov v osobných počítačoch, serveroch a vstavaných zariadeniach. OpenCL zvyšuje rýchlosť a odozvu širokého spektra aplikácií od herných až po medicínske [45].

Táto architektúra je veľmi blízka architektúre CUDA. CUDA device je postavený okolo škáľovateľného poľa multivláknových stream-ových multiprocesorov. Multiprocesor je ekvivalent výpočtovej jednotky OpenCL.

Multiprocesor vykonáva vlákno architekrúry CUDA pre každú OpenCL pracovnú jednotku (work-item) a blok vlákien pre každú OpenCL pracovnú skupinu (work-group). Kernel je spúšťaný nad OpenCL NDRange mriežkou blokov vlákien. Ako vidieť na obrázku 5.5, každý blok vlákien, ktoré spúšťa kernel, je unikátne identifikovaný svojím work-group ID a každé vlákno svojím global ID alebo kombináciou svojho local ID a work-group ID [7].



Obrázok 5.5: **Mriežka blokov vlákien** - Kernel je spúšťaný nad NDRange mriežkou blokov vlákien [7]

5.4.1 SIMT architektúra a warp

Multiprocesor je vytvorený na spúšťanie stoviek vlákien konkurentne. Pre správu takéhoto veľkého množstva vlákien používa architektúru *SIMT (Single-Instruction, Multiple-Thread)*. Spracovanie inštrukcií je zreťazené k využitiu paralelizmu v rámci jediného vlákna na inštrukčnej úrovni rovnako ako aj rozsiahly paralelizmus na úrovni vlákien [6].

Multiprocesor vytvára, spravuje, plánuje a spúšťa vlákna v skupinách 32 paralelných vlákien nazývaných *warpy*. Individuálne vlákna skladajúce warp sa spúšťajú spolu na rovnakej programovej adrese, ale majú svoj vlastný čítač inštrukčnej adresy a stav registra a teda môžu sa nezávisle vetviť a spúšťať. *Half-warp* je buď prvá alebo druhá polovica warpu. *Quarter-warp* je buď prvá, druhá, tretia alebo štvrtá štvrtina warpu [6].

Keď je multiprocesoru daný jeden alebo viac blokov vlákien na vykonanie, rozdelí ich do warpov, ktoré budú naplánované na spustenie plánovačom warpov. Spôsob, akým je blok rozdelený do warpov, je vždy rovnaký; každý warp obsahuje vlákna po sebe nasledujúce, zvyšujúce ID vlákna, kde prvý warp obsahuje vlákno 0. Warp spúšťa jednu spoločnú inštrukciu naraz, takže plná efektivita je dosiahnutá, keď všetkých 32 vlákien warpu majú

rovnakú vykonávanú cestu. Ak sa vlákna warpu oddelia pomocou podmieneného skoku závislom na dátach, warp serializuje beh takýchto vetiev s tým, že zastaví vlákna, ktoré do týchto vetiev nevstúpia a keď sa všetky vetvy vykonajú, vlákna konvergujú späť do rovnakej vykonávanej cesty. Divergencia vetvení sa objavuje iba vrámci jedného warpu; rôzne warpy sa vykonávajú nezávisle, nezáležiac na tom, či vykonávajú rovnakú alebo rôznu cestu kódu [7].

SIMT architektúra je príbuzná vektorovej organizácii SIMD (viď 5.2.1) v tom, že jedna inštrukcia kontroluje viaceré procesné elementy. Kľúčový rozdiel je v tom, že vektorová organizácia SIMD exponuje šírku SIMD software-u, zatial čo SIMT inštrukcie špecifikujú správanie spúšťania a vetvenia jediného vlákna. V porovnaní so SIMD vektorovými strojmi SIMT dovoľuje programátorom písať paralelný kód na úrovni vlákien pre nezávislé, skalárne vlákna tak, ako aj kód dátového paralelizmu pre koordinované vlákna. Pre účel správnosti kódu môže programátor ignorovať SIMT správanie, ale zamedzením zbytočného vetvenia v kóde sa dá dosiahnuť markantný nárast výkonu. V praxi je toto analogické roli cache v tradičnom kóde: veľkosť cache sa dá jednoducho ignorovať, keď sa vytvára správna aplikácia, ale musí byť braná do úvahy v štruktúre kódu, ak sa vytvára aplikácia pre dosiahnutie čo najlepšieho výkonu.

Ak neatomická inštrukcia vykonávaná warpom zapíše na rovnaké miesto v globálnej alebo zdielanej pamäti viac ako jedným vláknom warpu, počet serializovaných zápisov, ktoré sa v tomto mieste objavia, závisí na výpočtovej schopnosti zariadenia a ktoré vlákno vykoná posledný zápis, je nedefinované.

Ak atomická inštrukcia vykonávaná warpom číta, mení a zapisuje na rovnaké miesto v globálnej pamäti pre viac ako jedno vlákno warpu, každé čítanie, modifikovanie a zápis na toto miesto sa vykoná a všetky sú vlákna serializované, ale poradie, v akom sú vykonávané, je nedefinované [7].

5.4.2 Výpočtová schopnosť

Výpočtová schopnosť zariadenia device je definovaná hlavným a vedľajším revíznym číslom. Zariadenia device s rovnakým hlavným revíznym číslom majú rovnakú architektúru jadra. Vedľajšie revízne číslo zodpovedá úpravám a zlepšeniam architektúry jadra, prinášajúc nové možnosti.

Výpočtová schopnosť 1.x

Architektúra:

Pre zariadenia výpočtovej schopnosti 1.x, multiprocesor obsahuje:

- 8 CUDA jadier pre celočíselnú aritmetiku a operácie s pohyblivou rádovou čiarkou s jednoduchou presnosťou
- 1 jednotku pohyblivej rádovej čiarky s dvojitou presnosťou pre aritmetické operácie s pohyblivou rádovou čiarkou s dvojitou presnosťou
- 2 špeciálne funkčné jednotky (SFU) pre zvláštne funkcie s pohyblivou rádovou čiarkou s jednoduchou presnosťou (tieto jednotky zvládnu vykonávať násobenie desatinných čísel s jednoduchou presnosťou)
- 1 warp plánovač [7]

Pre spustenie inštrukcie všetkých vlákien warpu musí warp plánovač vykonať:

- 4 takty pre aritmetickú inštrukciu celého čísla alebo čísla s pohyblivou rádovou čiarkou s jednoduchou presnosťou
- 32 taktov pre aritmetické operácie s pohyblivou rádovou čiarkou s dvojitou presnosťou
- 16 taktov pre zvláštne operácie čísel s pohyblivou rádovou čiarkou s jednoduchou presnosťou

Multiprocesor má taktiež read-only konštantnú cache, ktorá je zdieľaná všetkými funkčnými jednotkami a urýchľuje čítania z konštantnej pamäte, ktorá je uložená v pamäti zariadenia [7].

Multiprocesory sú zgrupované v *Texture Processor Cluster-och (TPC)*. Počet multiprocesorov na TPC je:

- 2 pre zariadenia výpočtovej schopnosti 1.0 a 1.1,
- 3 pre zariadenia výpočtovej schopnosti 1.2 a 1.3.

Kgždé TPC má read-only textúrovaciu cache, ktorá je zdieľaná medzi všetkými multiprocesormi a urýchľuje čítanie z textúrovacej pamäti, ktorá sa nachádza v pamäti zariadenia [7].

Lokálna a globálna pamäť sa nachádza v pamäti zariadenia a nie je cache-ovaná.

Globálna pamäť

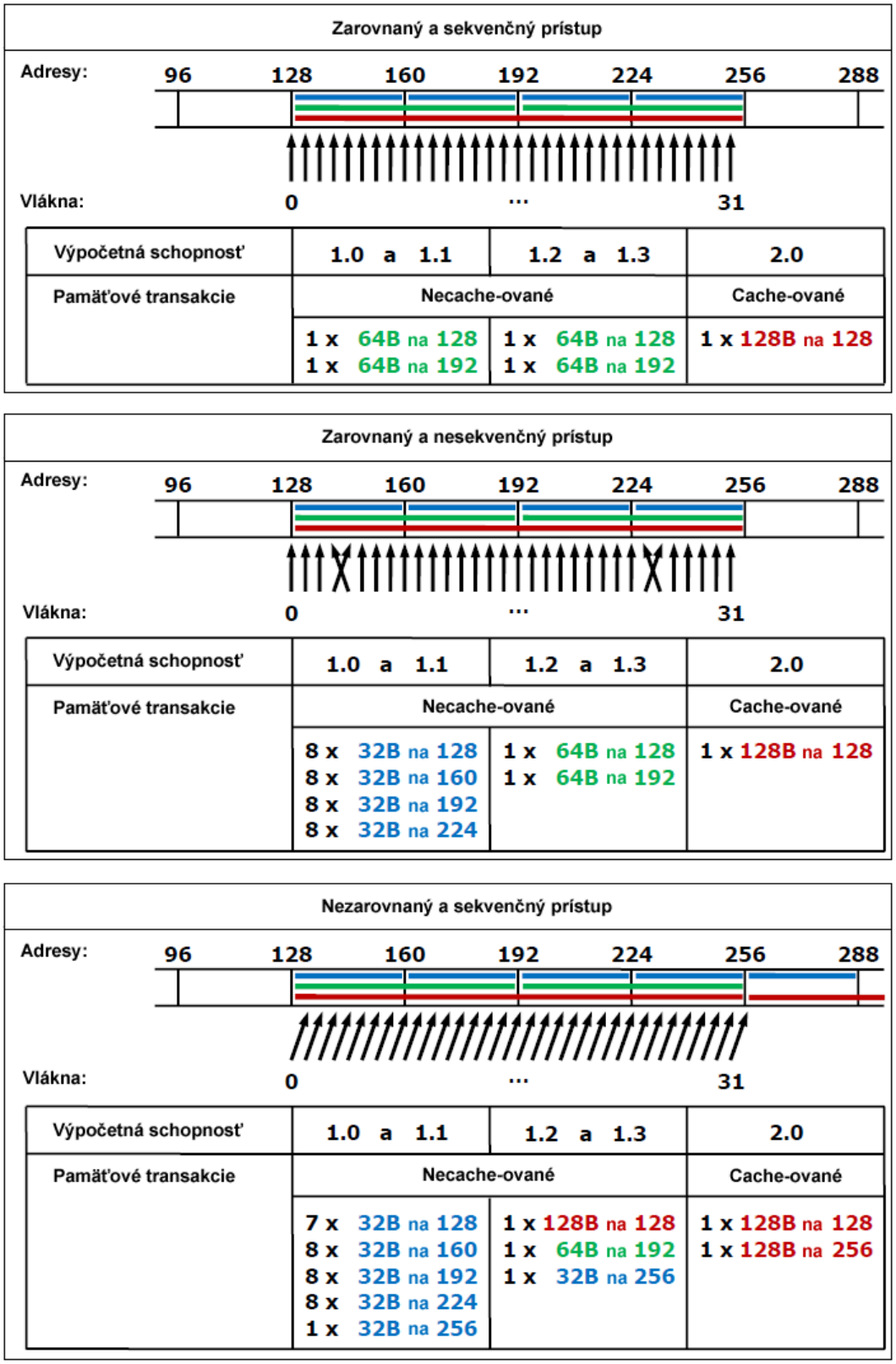
Požiadavka do globálnej pamäti warpu je rozdelená do dvoch pamäťových požiadaviek, jedna pre každý half-warp, ktoré sú vydané nezávisle na sebe. Obrázok 5.6 ukazuje niektoré prípady prístupu do globálnej pamäti a transakcie, ktoré zodpovedajú danej výpočtovej schopnosti.

Zariadenia výpočtovej schopnosti 1.0 a 1.1:

Pre zjednotený prístup do pamäti musí pamäťová požiadavka pre half-warp spĺňať podmienky:

- Veľkosť slov, ku ktorým chcú pristúpiť vlákna, musí byť 4, 8 alebo 16 bytov;
- Ak je táto veľkosť:
  - 4, všetkých 16 slov musí ležať v rovnakom 64-bytovom segmente,
  - 8, všetkých 16 slov musí ležať v rovnakom 128-bytovom segmente,
  - 16, prvých 8 slov musí ležať v rovnakom 128-bytovom segmente a posledných 8 slov v nasledujúcom 128-bytovom segmente;
- Vlákna musia pristupovať k slovám v sekvenciách: k-te vlákno v half-warpe musí pristupovať ku k-temu slovu.

Ak half-warp spĺňa tieto podmienky, 64-bytová transakcia, 128-bytová transakcia alebo dve 128-bytové transakcie sú vykonané, ak veľkosť pristupovaných slov vláknom je 4, 8 alebo 16, v tomto poradí. Zjednotený prístup je dosiahnutý, aj ak sa warp vetví a teda existujú vlákna, ktoré sú pozastavené a nepristupujú do pamäte [7].



Obrázok 5.6: Príklady prístupov do globálnej pamäti warpu, 4-bytové slová na vlákno a prislúchajúce pamäťové transakcie v závislosti od výpočtovej schopnosti [6]

Ak half-warp nespĺňa tieto požiadavky, je vykonaných 16 rôznych 32-bytových transakcií.

Zariadenia výpočtovej schopnosti 1.2 a 1.3:

Vlákna môžu pristupovať k slovám v rôznom poradí vrátane rovnakých slov a je vykonaná jediná pamäťová transakcia pre každý segment adresovaný half-warpom. Toto je veľký rozdiel v porovnaní so zariadeniami s nižšou výpočtovou schopnosťou, ktorých vlákna musia pristupovať k slovám v sekvenciách a zjednotený prístup nastáva, iba ak half-warp adresuje jediný segment [7].

Presnejšie, nasledujúci protokol sa používa pre zistenie pamäťových transakcií potrebných k obslúženiu všetkých vlákien v half-warpe:

- Nájsť pamäťový segment, ktorý obsahuje požadovanú adresu aktívnym vláknom s najnižším ID vlákna. Veľkosť segmentu závisí od veľkosti pristupovaného slova vláknom:
  - 32 bytov pre 1-bytové slová,
  - 64 bytov pre 2-bytové slová,
  - 128 bytov pre 4-, 8- a 16-bytové slová.
- Nájsť všetky ostatné aktívne vlákna, ktorých požadovaná adresa sa nachádza v rovnakom segmente.
- Redukovať veľkosť transakcie, ak je to možné:
  - Ak je veľkosť transakcie 128 bytov a iba nižšia alebo vyššia polovica je použitá, redukuje sa veľkosť transakcie na 64 bytov;
  - Ak je veľkosť transakcie 64 bytov (originálne alebo po redukcie zo 128 bytov) a iba nižšia alebo vyššia polovica je použitá, redukuje sa veľkosť transakcie na 32 bytov.
- Vykonať transakciu a označiť obslúžené vlákno ako neaktívne.
- Opakovať, kým všetky vlákna v half-warpe nie sú obslúžené [7].

Zdieľaná pamäť

Zdieľaná pamäť má 16 baniek, ktoré sú organizované tak, že nasledujúcich 32 bitov slova sú priradené do nasledujúcej banke, t.j. sú prekladané. Každá banka má šírku pásma 32 bitov na 2 takty.

Požiadavka do zdieľanej pamäti warpu je rozdelená do dvoch pamäťových požiadaviek, jedna pre každý half-warp a sú vykonané nezávisle na sebe. Dôsledkom toho je, že nemôže nastať bankový konflikt medzi vláknom patriacim do prvej polovice warpu a vláknom patriacim do druhej polovice toho istého warpu.

Ak neatomická inštrukcia spúšťaná warpom zapíše na rovnakú pozíciu v zdieľanej pamäti viac ako jedným vláknom warpu, iba jedno vlákno half-warpu vykoná zápis a je nedefinované, ktoré vlákno vykoná zápis ako posledné [7].

Zariadenia výpočtovej schopnosti 2.x

Multiprocesor zariadení výpočtovej schopnosti 2.x obsahuje:

- zariadenia výpočtovej schopnosti 2.0:

  - 32 CUDA jadier pre operácie celočíselnej a desatinnej aritmetiky,
  - 4 špeciálne funkčné jednotky pre zvláštne funkcie s desatinnými číslami jednoduchej presnosti

- zariadenia výpočtovej schopnosti 2.1:
  - 48 CUDA jadier pre operácie celočíselnej a desatinnej aritmetiky,
  - 8 špeciálne funkčné jednotky pre zvláštne funkcie s desatinnými číslami jednoduchej presnosti

- 2 warp plánovače

Pri každom spustení inštrukcie každý plánovač musí spustiť:

- 1 inštrukciu pre zariadenia výpočtovej schopnosti 2.0,
- 2 inštrukcie pre zariadenia výpočtovej schopnosti 2.1,

pre nejaký warp, ktorý je pripravený na vykonanie. Prvý plánovač má na starosti warpy s nepárnymi ID a druhý plánovač má na starosti warpy s párnymi ID. Keď plánovač spustí inštrukciu s desatinnými číslami s dvojitou presnosťou, druhý plánovač nemôže spustiť ďalšiu inštrukciu.

Warp plánovač môže spustiť inštrukciu iba polovici CUDA jadier. Aby bolo možné spustiť inštrukciu pre všetky vlákna warpu, warp plánovač musí spustiť inštrukciu počas dvoch taktov pre inštrukciu celočíselnej alebo desatinnej aritmetiky.

Multiprocesor má taktiež read-only uniformnú cache, ktorá je zdieľaná medzi všetkými funkčnými jednotkami a urýchľuje čítania z priestoru konštantnej pamäti, ktorá je uložená v pamäti zariadenia.

Nachádza sa tu L1 cache pre každý multiprocesor a L2 cache zdieľaná všetkými multiprocesormi, ktoré sú obe používané pre cache-ovanie prístupov do lokálnej alebo globálnej pamäti, vrátane dočasných registrov. Spravánie cache-e (napr. či sú čítania cache-ované aj v L1, aj v L2 alebo iba v L2) môže byť čiastočne konfigurované použitím modifikátorov pri load / store inštrukciách.

Rovnaká on-chip pamäť je použitá pre obe L1 a zdieľanú pamäť: môže byť konfigurovaná ako 48 kB zdieľanej pamäti a 16 kB L1 alebo 16 kB zdieľanej pamäti a 48 kB L1 cache [7].

5.4.3 Hardware-ový multithreading

Exekučný kontext (programové čítače, registre atď.) je pre každý warp spúšťaný multiprocesorom zachovávaný na čipe počas celej životnosti warpu. Prepínanie z jedného exekučného kontextu na druhý teda nič nestojí a v čase spúšťania každej inštrukcie vyberie warp plánovač warp, ktorý má vlákna pripravené na spustenie (aktívne vlákna) a spúšťa nasledujúcu inštrukciu týmto vláknam.

Podrobnejšie, každý multiprocesor má množinu 32 bitových registrov, ktoré sú rozdelené medzi warpy a paralelnú cache dát alebo zdieľanú pamäť, ktorá je rozdelená medzi bloky vlákien a použitá na implementáciu lokálnej pamäti OpenCL.

Počet blokov a warpov, ktoré môžu spočívať a byť spolu spracovánané na multiprocesore pre daný kernel, závisí na množstve registrov a zdieľanej pamäti použitej kernelom a množstvom registrov a zdieľanej pamäti dostupnej na multiprocesore. Existuje takisto aj maximálne množstvo použitých blokov a maximálne množstvo warpov na multiprocesor. Tieto limity, rovnako ako množstvo registrov a zdieľanej pamäti dostupnej na multiprocesore sú funkciou výpočtovej schopnosti zariadenia a môžeme ich vidieť v tabuľke 5.1.

	Výpočtová schopnosť				
Technická špecifikácia	**1.0**	**1.1**	**1.2**	**1.3**	**2.0**
Maximálny x-ový alebo y-ový rozmer mriežky blokov vlákien	65535				
Maximálny počet vlákien v bloku	512				1024
Maximálny x-ový alebo y-ový rozmer bloku	512				1024
Maximálny z-ový rozmer bloku	64				
Veľkosť warpu	32				
Maximálny počet blokov na multiprocesor	8				
Maximálny počet warpov na multiprocesor	24		32		48
Maximálny počet vlákien na multiprocesor	768		1024		1536
Počet 32 bitových registrov na multiprocesor	8 K		16 K		32 K
Maximálne množstvo zdieľanej pamäti na multiprocesor	16 kB				48 kB
Počet zdieľaných pamäťových baniek	16				32
Množstvo lokálnej pamäti na vlákno	16 kB				512 kB
Veľkosť konštantnej pamäti	64 kB				
Maximálne množstvo inštrukcií na kernel	2 milióny				

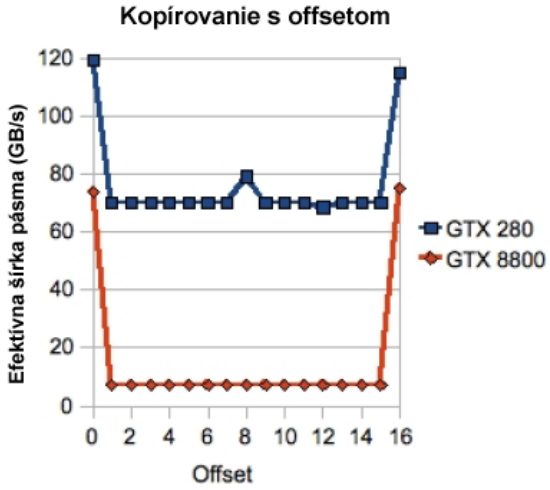
Tabuľka 5.1: Technická špecifikácia architektúr s výpočtovou schopnosťou 1.0 - 2.0

5.4.4 OpenCL optimalizácia

Optimalizácia pamäti

Pri optimalizácii pamäťových operácií treba dbať na 3 základné veci:

- Minimalizovať dátové prenosy medzi zariadením host a device:
  - Dátový prenos medzi zariadením host a device má oveľa menšiu šírku pásma ako prístup do globálnej pamäti - 8 GB/s (PCI-Express, x16 Gen2) vs 141 GB/s (GTX 280),
  - Minimalizovanie prenosu - preniesť kód CPU na GPU aj v prípade, že neprináša vyšší výkon, pokiaľ zníži prenos dát,
  - Zgrupovanie prenosu - jeden veľký prenos je oveľa lepši ako viac malých: latencia cca 10ms.
- Používať zjednotený prístup do globálnej pamäti - latencia globálnej pamäti je 400-800 cyklov. Prístup do globálnej pamäti vlákien half-warpu môže byť zjednotený do jednej transakcie slova veľkosti 8, 16, 32, 64 bitov alebo dvoch transakcií slova veľkosti 128 bitov (viď obr. 5.6). Následok nezarovnaného prístupu vidieť na obr. 5.7. Pri nezarovnanom prístupe sa na karte GTX280 (výpočtová schopnosť 1.3) zníži prenos 1.7-násobne a na karte GTX 8800 (výpočtová schopnosť 1.0) 8-násobne.
- Používať lokálnu (zdieľanú) pamäť ako cache
  - latencia je 100x menšia oproti globálnej pamäti,
  - vlákna dokážu spolupracovať pomocou lokálnej pamäti (vrámci bloku),
  - ukladanie dát do lokálnej pamäti pre zníženie prístupu do globálnej pamäti,



Obrázok 5.7: Príklad nezarovnaného prístupu do pamäti a jeho následok [46]

– používanie lokálnej pamäti pre odstránenie nezarovnaných prístupov do globálnej pamäti [46].

Aritmetické inštrukcie

- Celočíselné sčítavanie, bitový posun, minimum, maximum: 4 cykly (Tesla), 2 cykly (Fermi)
- Celočíselné násobenie 16 cyklov (Tesla), 2 cykly (Fermi)
- 32-bitové sčítanie, násobenie, mad, minimum, maximum s pohyblivou rádovou čiarkou: 4 cykly (Tesla), 2 cykly (Fermi)
- 64-bitové sčítanie, násobenie, mad s pohyblivou rádovou čiarkou: 32 cyklov (Tesla), 2 cykly (Fermi)
- Delenie a modulo (zvyšok po delení) sú drahé operácie
  - pri delení 2^n používať bitový posun: » n
  - pri modulo 2^n používať bitový súčin: & $(2^n - 1)$
- Zabrániť automaticému prevodu čísel double na float [46]

Kapitola 6

Návrh aplikácie a použitá konfigurácia

6.1 Výber aplikačných rozhraní, programovacieho jazyka a zariadenia GPU

Zadanie tejto práce je koncipované pre použitie multiplatformného štandardu ***OpenCL***. Vypracovať túto prácu som sa rozhodol v operačnom systéme ***Windows*** a vývojovom prostredí ***Visual Studio***.

Keďže device kód OpenCL je postavený na jazyku C a štandarde C99, programovací jazyk ***C/C++*** bol zvolený aj pre programovanie kódu na CPU. Existujú aj ďalšie možnosti, medzi ktoré patria napríklad programovacie jazyky Java (JOCL), Python (PyOpenCL) a iné, ale s programovaním v jazyku C mám najväčšie skúsenosti.

Pre spustenie OpenCL kódu na grafickej karte je nutné použiť minimálne kartu od radu GeForce 8 s označením G80, pretože toto sú prvé karty, ktoré podporujú unifikovaný programový model CUDA. Aby bolo možné spustiť OpenCL kód aj na CPU, musí sa jednať o procesor firmy Intel a medzi podporované procesory patria:

Mobilné a stolné produkty

- Procesory Intel Core i7 Extreme Edition
- Procesory Intel Core i7
- Procesory Intel Core i5
- Procesory Intel Core i3
- Intel Core 2 Extreme Processor, rad 9000
- Procesory Intel Core 2 Quad
- Intel Core 2 Duo Procesor, rad 8000
- Intel Core 2 Duo Procesor E7200

Serverové produkty:

- Procesory Intel Xeon, rady 7500, 7400

- Procesory Intel Xeon, rad 5500
- Quad-Core Intel Xeon Procesor rady 5400, 3300
- Dual-Core Intel Xeon Procesor, rady 5200, 3100

Podporované operačné systémy:

- Microsoft Windows 7 (32-bitová verzia)
- Microsoft Windows 7 (64-bitová verzia)
- Microsoft Windows Vista (32-bitová verzia)
- Microsoft Windows Vista (64-bitová verzia)
- Novell SUSE Linux Enterprise Server 11 SP1 (64-bitová verzia)
- Red Hat Enterprise Linux 6 (64-bitová verzia)

Aplikácia bola vyvíjaná a testovaná na grafickej karte ***NVIDIA GeForce GTX 280*** s výpočtovou schopnosťou 1.3 (viď kapitola 5.4.2). Testy boli prevádzané ďalej na grafickej karte NVIDIA GeForce GTX 480 s architektúrou Fermi a výpočtovou schopnosťou 2.0. Aby bolo možné porovnať výkon GPU oproti CPU, pre testovanie bol použitý štvorjadrový procesor Intel Core i7 920. V dobe vyvíjania aplikácie boli dostupné ovládače Intel OpenCL iba vo verzii alpha. V dobe písania tejto práce vyšli novšie beta ovládače, s ktorými aplikácia nebola vyvíjaná, ani testovaná.

Pre zobrazenie cestnej premávky bolo použité prostredie ***OpenGL***. Vizualizácia je jednoduchá 2D mapa s rozmiestnenými cestami a po každom behu kernelu sa na tieto cesty „poukladajú“ autá na ich novej pozícii po vypočítanom diskrétnom kroku celulárneho automatu.

6.2 Návrh aplikácie

Ako každý problém, aj tento je možné riešiť rôznymi spôsobmi. Niektoré prístupy sú viac, iné menej efektívne a vhodné. Pri návrhu tejto aplikácie bola snaha zvoliť prístup čo najväčšej jednoduchosti. Návrh aplikácie prebiehal v niekoľkých fázach, ktoré sú všetky úzko prepojené:

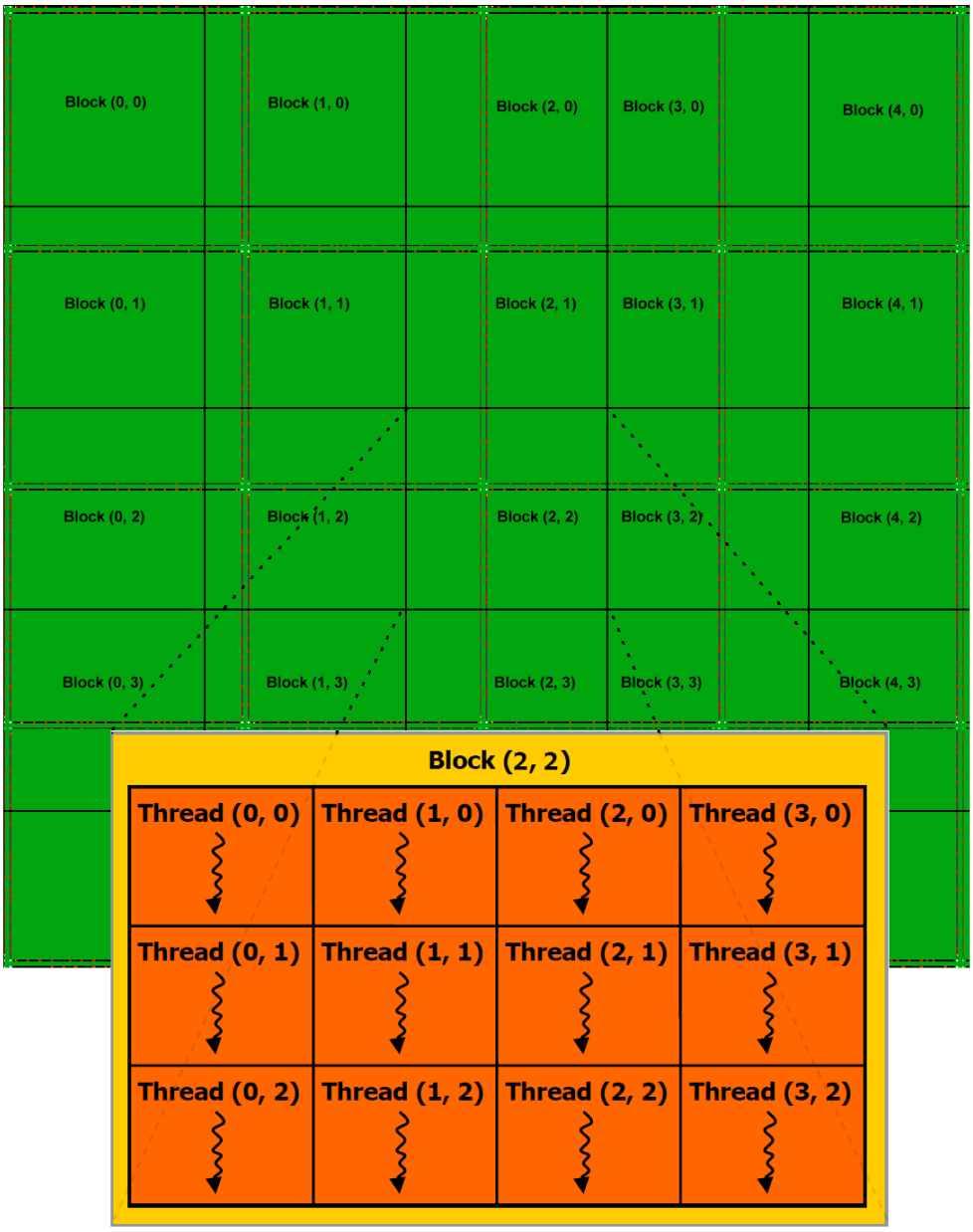
1. Ako prvé bolo potrebné si povedať, čo všetko bude daný model schopný robiť, ako sa bude správať. Keďže bol kladený dôraz na jednoduchosť modelu a jeho reprezentácie, jedná sa o veľkú mriežku ciest a križovatiek s určenou hlavnou a vedľajšou cestou. Hranice mapy sú ukončené cyklickými hraničnými podmienkami podobne, ako to bolo popísané v kapitole 3.2.
2. S prvým krokom prichádza návrh reprezentácie dát, viď kapitola 7.
3. S predchádzajúcim krokom súvisí okolie, aké sa bude používať pri výpočte nasledujúceho kroku celulárneho automatu. Mapa je rozdelená na kvadranty, ktorých je počas behu celého programu konštantný počet. Každý kvadrant pokrýva konkrétnu časť mapy, autá do tohto kvadrantu vchádzajú a z neho vystupujú, čiže počet áut v kvadrante sa mení. Kvadrant obsahuje zoznam áut, ich počet a rýchlosti a ďalšie dáta, medzi ktoré patrí napríklad informácia, kam chce auto odbočiť na križovatke.

4. Ďalej bolo nutné zamyslieť sa, akým spôsobom bude možné rozdeliť dáta pre paralelné spracovanie. Keďže je mapa rozdelená na kvadranty a tie obsahujú jednotlivé autá, bol zvolený prístup taký, že každý kvadrant sa spracúva vo vlastnom bloku (work-group) kernelu a každé auto kvadrantu jedným vláknom (work-item), viď obr. 6.1.

5. Výpočty a kopírovania z/do globálnej pamäti bolo potrebné rozdeliť do viacerých spúšťaných kernelov z dôvodu, že synchronizácia globálnej pamäti na úrovni mriežky blokov neexistuje. To je spôsobené tým, že keď vlákno v rámci kernelu zapíše dáta do globálnej pamäti, tá nebude dostupná vláknam bloku, ktoré boli spúšťané pred týmito blokmi, pretože ich vykonávanie už bolo ukončené. Jedinou zaručenou cestou, ako „synchronizovať“ globálnu pamäť, je zastaviť súčasný kernel a spustiť nový.

6. Poslednou súčasťou návrhu bola vizualizácia. Po každom behu kernelov potrebných na výpočet nových pozícií vozidiel na mape sa na základe novovzniknutých dát vykreslí pomocou OpenGL cestná sieť spolu s komunikáciami a vozidlami.

Priemerná dĺžka osobných vozidiel sa pohybuje od 4 do 4.5 metra. Veľkosť jednej bunky sa v aplikácii rovná piatim metrom. Toto je dĺžka, do ktorej sa zmestí jedno vozidlo a ešte sa na oboch koncoch auta nachádza miesto, takže pokiaľ sa nachádzajú autá v dvoch susedných bunkách, majú medzi sebou rozostup. Pokiaľ sa autá pohybujú po celočíselnom násobku počtu buniek, znamená to, že ich rýchlosť je násobkom 5 (pretože veľkosť bunky je 5m) a ak si časový interval medzi dvomi stavmi celulárneho automatu zvolíme ako sekunda, zvyšovanie a znižovanie rýchlosti áut sa pohybuje po 5 $m * s^{-2}$.



Obrázok 6.1: Rozdelenie hierarchie kvadrantov do mriežky blokov vlákien. Jeden konkrétny blok, napr. [2, 2] predstavuje kvadrant na rovnakej pozícii [2, 2]

Kapitola 7

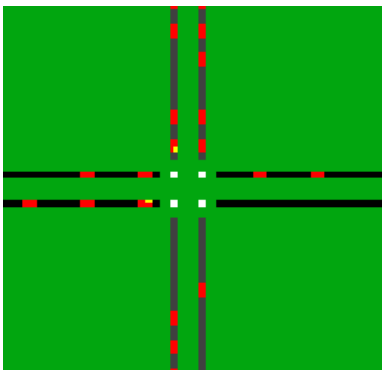
Dátová reprezentácia

Celá dátová reprezentácia sa skladá z jednorozmerných polí bez použitia zložitejších štruktúr (kvôli možnosti cache-ovania grafickej karty) a ich veľkosť závisí od použitia a určenia v programe. Z hľadiska optimalizácie sa používajú výlučne rozmery polí tak, aby ich veľkosť bola mocninou čísla 2 (ako bolo popísané v kapitole 5.4.4) pre možnosť rýchleho zisťovania indexov a možnosti obísť časovo náročné operácie delenia.

Mapa, kde sa autá nachádzajú a po ktorej sa pohybujú, má štvorcový charakter. Rozmer mapy sa pohybuje od 1024x1024 buniek do 2048x2048 buniek celulárneho automatu. Sieť ciest a križovatiek tvorí mriežku, ktorej veľkosť môže užívateľ pomocou vstupných parametrov programu meniť. V mape sa nachádza obojsmerná premávka (viď obr. 7.1).

V dátovom poli *mapa* sa však záznamy o autách samotných a ich premávke nenachádzajú. K tomuto účelu slúžia ďalšie polia, a to polia kvadrantov. Tie boli zavedené z dôvodu, že by bolo výkonovo náročné zisťovať sekvenčným prehľadávaním mapy, kde všade na mape sa autá nachádzajú (keďže prevažná časť mapy ani neobsahuje samotnú cestnú infraštruktúru, na ktorej sa nachádzajú autá). Išlo by o zbytočné plytvanie zdrojmi. Jediný problém, ktorý v návrhu s kvadrantmi nastáva, je ten, že vozidlá vrámci jedného kvadrantu sú navzájom pomiešané a nie je možné povedať, kde v kvadrante sa nachádza susedné vozidlo aktuálne skúmaného vozidla. Je teda nutné prejsť v cykle všetky autá kvadrantu, ale pokiaľ sú veľkosti kvadrantov vhode zvolené a existuje rozumný maximálny počet vozidiel v kvadrante, nepredstavuje toto veľký problém.

Každý kvadrant obsahuje informáciu o tom, koľko vozidiel sa v ňom nachádza, ďalej



Obrázok 7.1: Grafická ukážka obojsmernej premávky aplikácie

dáta o týchto vozidlách - na akej aktuálnej pozícii mapy sa nachádzajú, akú majú aktuálnu rýchlosť, či chcú odbočiť doprava, doľava, alebo ísť rovno a pod. Ďalej je daný maximálny počet vozidiel v kvadrante, ktoré nesmie byť prekročený (ak by sa tak stalo, boli by prepisované dáta okolitých kvadrantov). Takýto prípad však nemôže nastať, lebo kvadranty sú programom vytvárané tak, aby aj pri 100% zaplnení cesty autami maximálny počet vozidiel kvadrantu nemohol byť prekročený.

Pre účely „predávania“ vozidiel medzi kvadrantami existujú dočasné polia v globálnej pamäti, ktoré nesú informáciu o tom, koľko vozidiel sa má presunúť do ktorého kvadrantu (pri prekonávaní hraníc kvadrantu). Tieto dáta sa pripravujú počas behu prvého kernelu a beh druhého kernelu je vyhradený pre skopírovanie dočasných záznamov do kvadrantov v globálnej pamäti.

V kerneloch sa ďalej používa generátor pseudonáhodných čísel. Ten je potrebný pre náhodné určenie, kam sa má auto vydať (odbočit vľavo, vpravo, alebo ísť rovno), keď príde na križovatku. Generátor čísel sa ešte využíva pri náhodnom spomaľovaní áut v situáciách, keď to premávka nevyžaduje, ale vodič nechcene (či už vplyvom vyrušovania zavineného samým sebou alebo cudziou osobou) spomalí. Semienka pre generátor pre každé vlákno (pre každé spracovávané auto vo všetkých kvadrantoch) sú po spustení programu vygenerované v kóde host na CPU a následne nahrané do globálnej pamäti zariadenia device.

Kapitola 8

Implementácia

Implementácia aplikácie sa delí na dve kľúčové časti. Je to programovanie na strane CPU (kód Host) a na strane GPU (kód Device).

8.1 Kód Host

8.1.1 Inicializácia a príprava dát

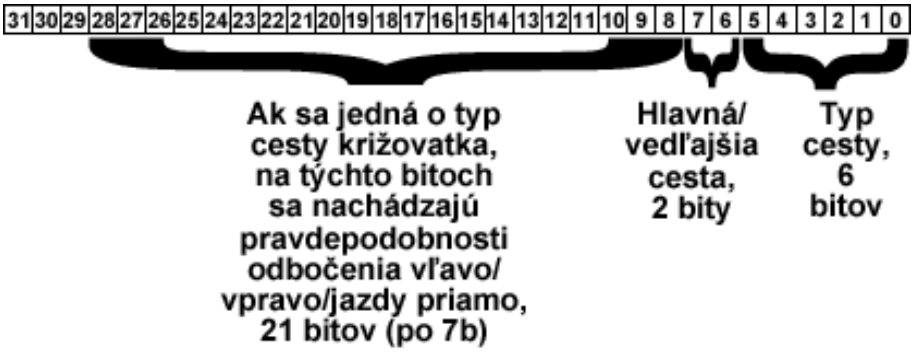
Získaním vstupných parametrov od užívateľa je možné inicializovať dáta, ktoré sa pripravujú iba raz na začiatku programu. Veľkosť bunky je konštantná, má veľkosť 5 metrov a nedá sa meniť. Každé auto sa nachádza vždy práve v jednej bunke.

Inicializácia prebieha naraz, v poli mapa sa nachádza informácia o tom, či sa v danej bunke nachádza cesta a ak áno, nastavia sa sem aj bity určujúce, ktorým zo štyroch smerov (zľava, zprava, zhora, zdola) sa budú autá na nej pohybovať. Pokiaľ sa nejedná o pruh cesty, ale o križovatku, pripravia sa aj bitové dáta s pravdepodobnosťami odbočenia auta do jednotlivých smerov. Toto všetko je obsiahnuté v jednej premennej typu int, ako zobrazuje obrázok 8.1.

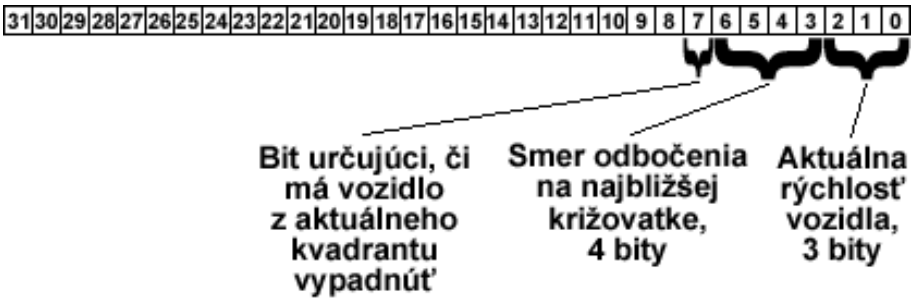
Do kvadrantov sa pri inicializácii vkladajú vozidlá podľa pravdepodobnosti hustoty vyťaženia komunikácie, ktorú je možné meniť parametrom pri spustení programu. Každé vozidlo má počiatočnú rýchlosť $v = 0[m*s^{-1}]$. Bitová reprezentácia vozidla zahŕňa jeho aktuálnu rýchlosť, kam má odbočiť na najbližšej križovatke a jeden bit je vyhradený pre určenie, či bolo auto „vylúčené“ z kvadrantu a presunuté do ďalšieho, susedného. Ak sa pri spracovaní kernelu 1 zistí výskyt takéhoto bitu, vozidlo sa v danom kvadrante nespracúva (podrobnejšie v 8.2.1). Rozloženie bitov vozidla vidieť na obrázku 8.2.

8.1.2 3 druhy kernelov

Pri posune vozidiel v mape často nastáva situácia, keď vozidlo vychádza z aktuálneho kvadrantu a presúva sa do vedľajšieho. Teoreticky (a mnohokrát aj prakticky) sa môže stať, že viacero kvadrantov „vypúšťa“ vozidlá do jedného, rovnakého kvadrantu. Situáciu znázorňuje obrázok 8.3. Z hľadiska GPU architektúry a návrhu aplikácie nie je možné zapísať do globálnej pamäti vláknami v rôznych blokoch (kvadrantoch) takúto situáciu do rovnakého kvadrantu bez nechceného prepisu a teda nechcenej straty dát. Takýto prípad sa rieši tak, že sa aktuálny kernel zastaví a spustí sa kernel nový, kde už budú aktuálne upravené dáta všetkých blokov predchádzajúceho kernelu. V prvom kerneli sa teda upravia hodnoty



Obrázok 8.1: Bitová reprezentácia bunky cesty

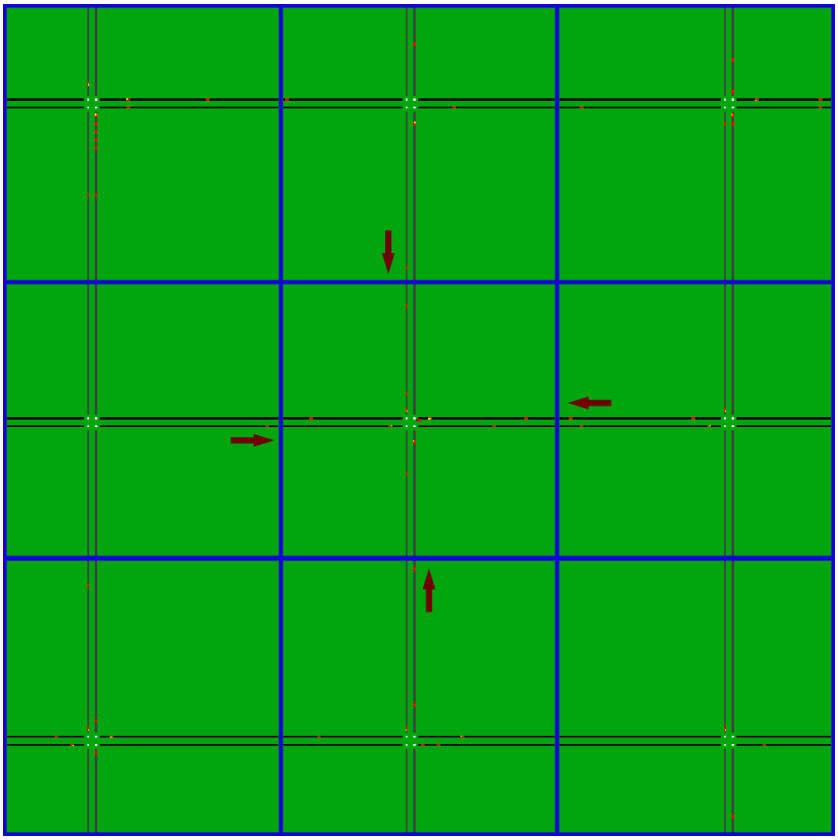


Obrázok 8.2: Bitová reprezentácia vozidla

rýchlosti a pozícií vozidiel a vozidlá, ktorých pohyb spôsobí opustenie kvadrantu, v ktorom sa nachádzajú, sú presunuté do dočasnej globálnej pamäti, ktorú má každý kvadrant vlastnú, takže nenastáva žiadne nežiadúce prepisovanie pamäti. Po skončení prvého kernelu zoberie druhý kernel všetky tieto „dočasné“ dáta a zapíše ich na príslušné miesta do globálnej pamäti kvadrantov jednoducho tak, že kvadrant (blok mriežky) si pozbiera dáta okolitých kvadrantov a zapíše k sebe.

Vozidlá, ktoré sa nachádzajú tesne pred križovatkou, sú spracúvané v treťom, poslednom kerneli zameranom na spracovanie križovatiek. Všetky tieto 3 kernely sú spúšťané po sebe sériovo (na kartách s čipom GT200 nie je možné spúšťať viac kernelov naraz, ale v tomto prípade by to ani nebolo žiadúce, pretože je potrebné, aby vykonávanie predošlých kernelov bolo ukončené v čase behu nasledujúceho kernelu) v cykle, ktorého počet opakovaní je možné zadať parametrom programu.

Na začiatku pri prvom behu sa za zdroj dát celulárneho automatu považujú nainicializované dáta. Tie sa nakopírujú do globálnej pamäti a zároveň sa vytvorí priestor v globálnej pamäti s rovnakou veľkosťou, akú majú vyhradené zdrojové dáta. Do týchto cieľových polí sú nahrávané údaje o novom stave celulárneho automatu (nové pozície áut, rýchlostí a pod.). Pri druhom behu dávky týchto troch kernelov sa za zdroj dát považujú cieľové dáta predchádzajúceho behu a cieľovým poľom sa stáva pamäť zdrojových dát predchádzajúceho behu. Toto sa celé opakuje dookola, kým beh programu nie je prerušený užívateľom alebo vstupnou podmienkou na maximálny počet iterácií.



Obrázok 8.3: Ukážka prechodu vozidiel z viacerých (bočných) kvadrantov do jedného (stredného)

8.1.3 Dodatočná úprava dát

Raz za niekoľko behov trojice kernelov sú zmenené pravdepodobnosti križovatiek, kam majú autá, prichádzajúce na tú-ktorú križovatku, odbočiť. Parameter, ako často sa majú pravdepodobnosti meniť, je možné zadať programu pri spustení. Pravdepodobnosti sa po ich reinicializovaní nahrajú opať do globálnej pamäti.

8.2 Kód Device

8.2.1 Algoritmus výpočtu nasledujúceho stavu celulárneho automatu

Celulárny automat je navrhnutý tak, aby pracoval na globálnej úrovni jednotlivých vozidiel v kvadrante. Každý kvadrant (blok mriežky kernelu) si do zdieľanej pamäti nakopíruje dáta prislúchajúce tomuto kvadrantu a dáta svojich okolitých kvadrantov tak, aby nebola prekročená maximálna veľkosť zdieľanej pamäti na multiprocesor (viď tabuľku 5.1). Každé vozidlo kvadrantu predstavuje jedno vlákno bloku. Keďže zdieľaná pamäť je prístupná všetkým vláknam v rámci bloku, využíva sa táto skutočnosť pri vykonávaní nasledujúceho algoritmu.

Najprv sa v rámci vlákna (a teda jedného konkrétneho vozidla) prechádzajú v cykle všetky vozidlá daného kvadrantu. Pokiaľ sa nachádza nejaké vozidlo pred skúmaným vo-

zidlom, táto informácia sa uloží do registrov vlákna spolu s jeho aktuálnou rýchlosťou, pozíciou a vzdialenosťou od skúmaného auta. Ak sa žiadne takéto auto v aktuálnom kvadrante nenašlo, prehľadáva sa susedný kvadrant, v ktorom by sa potenciálne mohlo nachádzať auto, ktoré je v nejakej vzdialenosti pred skúmaným autom. Ak sa už ani tu nič nenájde, nemá sa ďalej čo prehľadávať a dá sa z tohto vyvodiť záver, že vozidlo sa môže voľne pohnúť dopredu. Je však nutné ešte skontrolovať, či sa v blízkosti pohybu vozidla nenachádza križovatka.

V ďalšom kroku algoritmu kernelu je teda nutné prehľadať priestor pred vozidlom a hľadať cestu typu *križovatka*. Typ cesty sa prehľadáva do vzdialenosti $d = MAX\_SPEED * (MAX\_SPEED >> 1) + MAX\_SPEED$, kde MAX_SPEED je konštanta maximálnej rýchlosti a $MAX\_SPEED >> 1$ je bitový posun o 1 doprava, čiže sa vlastne jedná o celočíselné delenie číslom 2. Vzdialenosť d je najväčšia vzdialenosť, akú je auto schopné prejsť spomaľovaním z maximálnej možnej rýchlosti. Predpokladá sa spomaľovanie jednej bunky za časovú jednotku a teda spomalenie $a = -5m * s^{-2}$. Pokiaľ sa nejaká križovatka nájde v takomto okolí, zapíše sa informácia o jej pozícii do registra (lokálnej premennej) vlákna.

Predošlé súčasti algoritmu dávajú predstavu o tom, ako môžu vozidlá ďalej v mape postupovať. Na tomto mieste je možné vypočítať, aká bude nová hodnota rýchlosti vozidla v závislosti od hodnôt zistených do tohto bodu. Tá sa vypočíta buď na základe vzdialenosti od nasledujúceho auta alebo od toho, ako ďaleko sa nachádza križovatka. Vo funkcii pre zistenie novej rýchlosti sa zároveň zisťuje, či je v blízkosti križovatky a ak je, určí sa autu (ak mu ešte určené nebolo), kam má odbočiť. Toto sa nastaví náhodne na základe pravdepodobností pre rôzne smery danej križovatky.

V oblasti križovatky je maximálna povolená rýchlosť nižšia oproti zvyšným častiam cesty. Pokiaľ auto pokračuje na hlavnej ceste pred križovatkou priamo, nie je nutné, aby úplne zastalo (pokiaľ sa pred ním nenachádza iná prekážka). V ostatných prípadoch musia autá zastaviť a pozrieť sa, či nekrižujú cestu iným autám prechodom do iného pruhu križovatky.

V kerneli, ktorý spracúva križovatky, sa zistí, ktoré autá chcú kam odbočiť a na základe okolia križovatky sa zistí, či je takýto úkon možné urobiť, alebo sa musí čakať, kým sa ostatné pruhy uvoľnia. Tento kernel má iné rozloženie blokov ako predošlé dva. Líšia sa v tom, že kernel na križovatky má počet blokov určený počtom križovatiek na mape. Každú križovatku teda spracúva jeden blok mriežky kernelu, vždy jedným vláknom.

8.2.2 Využitie lokálnej (zdieľanej) pamäti

Lokálna pamäť sa využíva iba v prvom kerneli. Nahrávajú sa do nej údaje z globálnej pamäti (ktorá má 100x väčšiu latenciu, viď sekcia 5.4.4) aktuálne spracovávaného kernelu a okolitých kernelov. Zároveň sa do lokálnej pamäti ukladajú novozískané hodnoty a až nakoniec sú naraz všetky dáta prekopírované do globálnej pamäti, kde sa s nimi v ďalších kerneloch ďalej pracuje. Toto využitie lokálnej pamäti prináša niekoľkonásobný nárast výpočtového výkonu.

V ostatných dvoch kerneloch nebolo potrebné lokálnu pamäť použiť. Druhý kernel sa využíva iba na prekopírovanie údajov z dočasnej globálnej pamäti do globálnej pamäti kvadrantov. A tretí kvadrant si potrebuje prejsť dáta z globálnej pamäti iba raz, tým pádom je zbytočné, aby sa zapájala lokálna pamäť do tohto procesu.

8.2.3 Optimalizácie kódu

Ako už bolo v texte viackrát spomenuté, v kerneli sa využíva zdieľaná pamäť, do ktorej prístup zaberá oproti pamäti globálnej 100x menej času. Okrem využitia lokálnej pamäti boli použité mnohé doporučenia pre optimalizáciu GPU kódu.

Aby bolo možné vyhnúť sa častému používaniu operácie delenia kvôli zisťovaniu indexov a x, y súradníc, veľkosti polí a množstvo kvadrantov sú mocniny čísla 2. V takomto prípade je možné využiť namiesto delenia bitové posuny a bitový AND (viď sekcia 5.4.4).

Pri optimalizácii kódu sa dbalo na to, aby sa zbytočne nevetvil a nevykonával veľa takých operácií, kde nastáva divergencia a je nutné potom takéto časti serializovať. Toto sa zabezpečilo napríklad tým, že podmienky v cykloch závislé na dátach sa predpripravia dopredu v malých častiach kódu a potom nie je nutné vetviť veľké časti. Takýmto prístupom sa zmenšuje aj veľkosť kódu.

Kapitola 9

Dosiahnuté výsledky

Aplikácia bola testovaná s konfiguráciou zobrazenou v tabuľke 9.1 pre testovanie na grafickej karte GeForce GTX 280 a procesore Intel Core i7 920. Pre testovanie na grafickej karte GeForce GTX 480 (s architektúrou Fermi) bola použitá konfigurácia zobrazená v tabuľke 9.2.

Testovanie prebiehalo s rôznymi nastaveniami veľkostí mapy, hustoty premávky (množstvom vozidiel na komunikácii), vzdialenosti medzi križovatkami a pod. Testy boli prevádzané takým spôsobom, aby inicializácie zariadení nespôsobovali skreslenie výsledkov a teda prvých 1000 behov každej konfigurácie nebolo braných do úvahy a do výsledkov testov sa uvažovali až priemerné hodnoty behov 1001-2000.

Procesor Intel Core i7 bol testovaný multithread-ovým prístupom s využitím ovládača OpenCL firmy Intel. Boli teda využívané všetky 4 jadrá tohto procesora a naraz spúšťaných 8 vláken vďaka technológii hyperthreading.

V nasledujúcich sekciách tejto kapitoly budú testy postupne analyzované od najvšeobecnejšieho prístupu (porovnanie časov behov kernelov rôznych zariadení) až po najpodrobnejší pohľad (porovnanie urýchlení jednotlivých kernelov a pomer stráveného času v jednotlivých kerneloch rôznych zariadení).

9.1 Porovnanie rýchlosti výpočtu kernelov na CPU a GPU všeobecne

Po prvom pohľade na graf 9.1 je zrejmé, že sa podarilo urýchliť pomocou grafickej karty simuláciu dopravy navrhnutým algoritmom. Nárast výkonu môžeme vidieť na grafe 9.2. Zaujímavé je, že narastajúcim počtom vozidiel na ceste sa výkon grafickej karty v porovnaní s procesorom CPU znižuje.

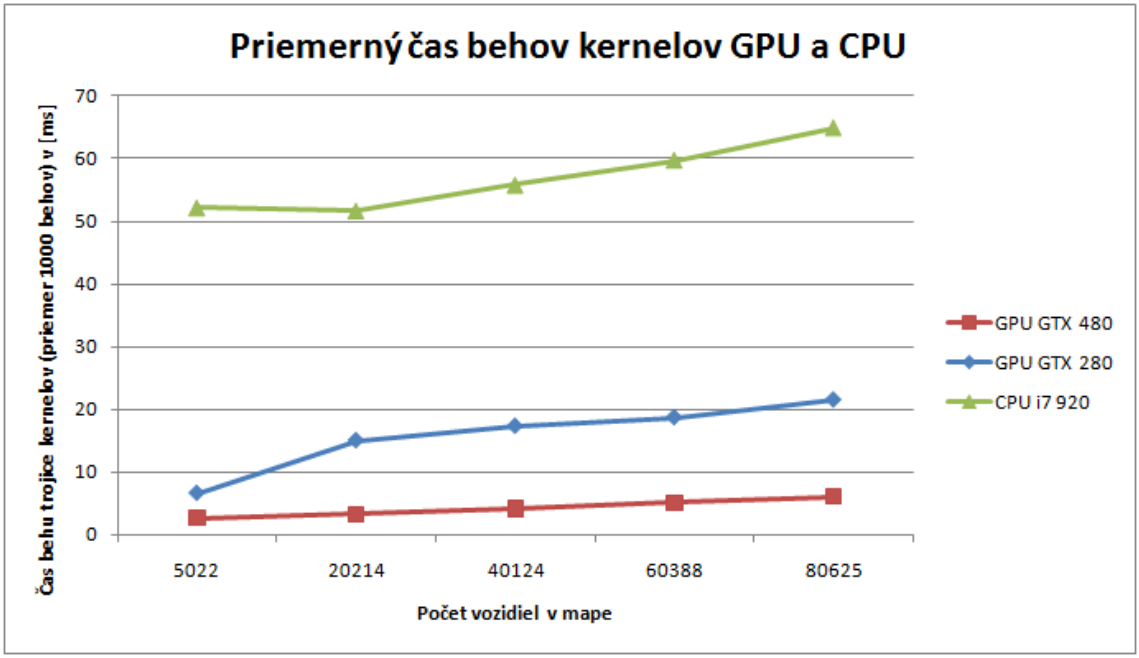
Pre porovnanie v grafe 9.3 vidíme veľkú mapu (2048x2048 buniek, 2021.76 km vozoviek) s hustotou premávky od 5% do 80%. Výsledky týchto grafov ukazujú, že nárast výkonu v zložitejšej dopravnej sieti je ešte väčší, ako v menšej sieti a s menším počtom vozidiel,

CPU	Intel Core i7 920, 4 jadrá, hyperthreading, 2.66 GHz
RAM	6 GB DDR3
GPU	NVIDIA GeForce GTX 280

Tabuľka 9.1: Použitá testovacia konfigurácia pre grafickú kartu GTX 280 a CPU Intel Core i7 920

CPU	Intel Core i7 920, 4 jadrá, hyperthreading, 2.66 GHz
RAM	6 GB DDR3
GPU	NVIDIA GeForce GTX 480

Tabuľka 9.2: Použitá testovacia konfigurácia pre grafickú kartu GTX 480



Obrázok 9.1: Veľkosť mapy 1024x1024 buniek, 505.44 km ciest

čo sa pôvodne z prvého grafu neočakávalo (viď graf 9.4). Tento vývoj môže byť spôsobený veľkým počtom procesných elementov a simultánne spúšťaných vláken grafickej karty oproti procesoru CPU.

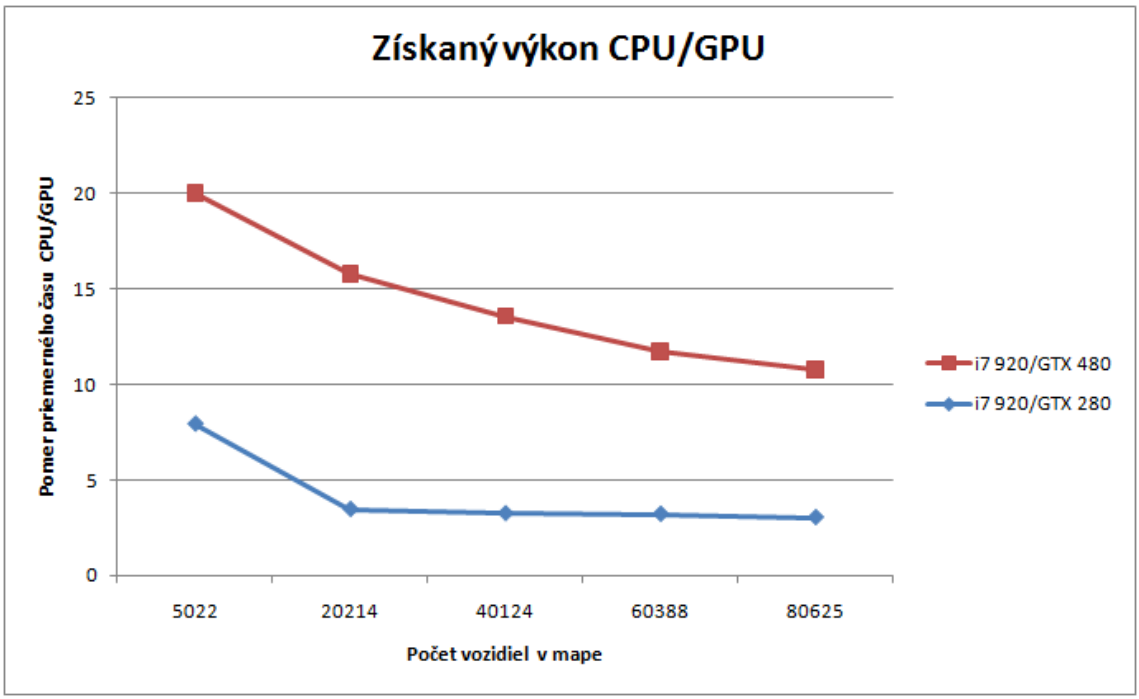
9.2 Porovnanie behov jednotlivých kernelov zariadení CPU a GPU

Tabuľka 9.3 približuje, čo sa v jednotlivých zariadeniach deje počas evolúcie celulárneho automatu. Väčší percentuálny rozptyl niektorých položiek v tabuľke (napr. GTX 280, kernel 1, 40-70 %) je spôsobený rôznym počtom vozidiel v mape a nutnosti počítania a zisťovania väčšieho okolia vozidiel.

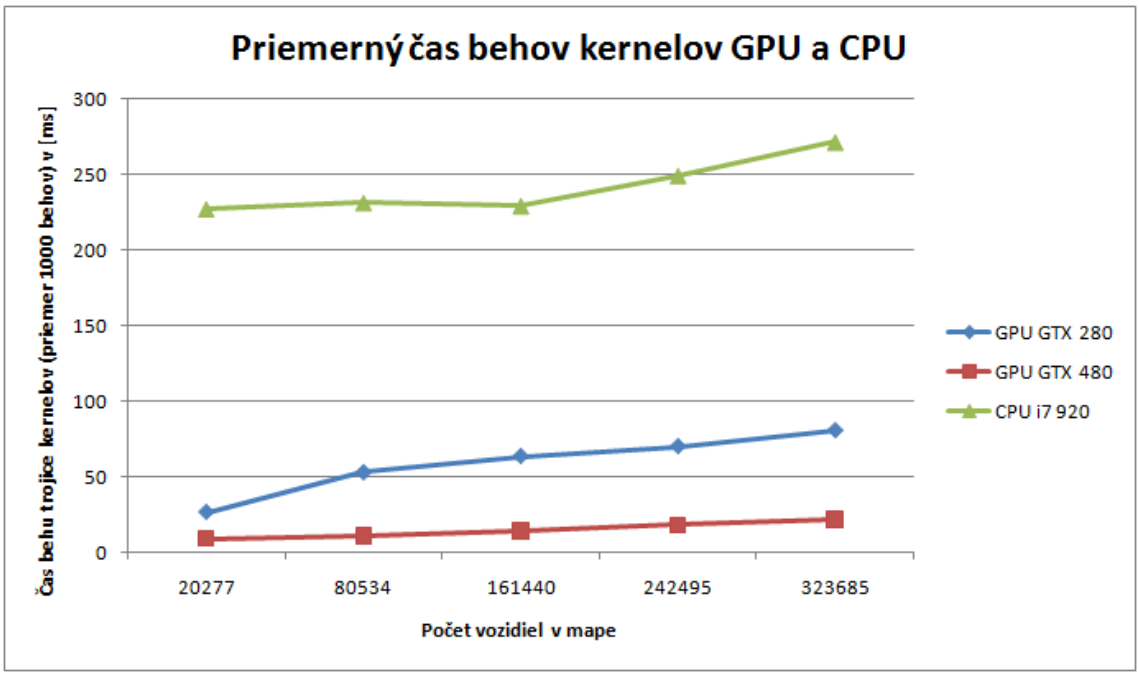
Čo je však zaujímavé na tejto tabuľke, je pomer času strávaného v kerneloch 2 a 3

	kernel 1	**kernel 2**	**kernel 3**
GTX 480	31-56%	4-17%	28-60%
GTX 280	40-70%	3.5-10%	15-54%
Intel Core i7	40-55%	43-58%	1-3%

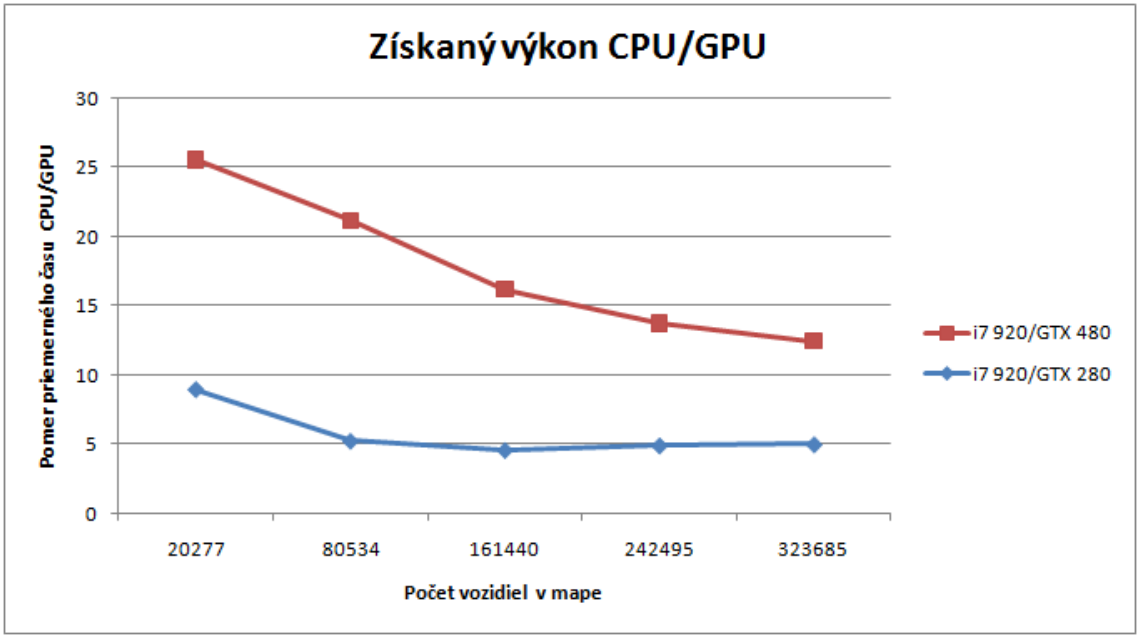
Tabuľka 9.3: Pomer tráveného času v jednotlivých kerneloch



Obrázok 9.2: Veľkosť mapy 1024x1024 buniek, 505.44 km ciest



Obrázok 9.3: Veľkosť mapy 2048x2048 buniek, 2021.76 km ciest



Obrázok 9.4: Veľkosť mapy 2048x2048 buniek, 2021.76 km ciest

zariadení CPU a GPU. Obe grafické karty trávia oveľa viac času v treťom kerneli (narozdiel od CPU, ktorý tento kernel využíva iba v malom zlomku celkového času kernelu), zatiaľ čo v druhom kerneli je pomer stráveného času oveľa menší oproti CPU. V druhom kerneli sa iba kopírujú dáta z dočasnej globálnej pamäti späť do globálnej pamäti kvadrantov. Prenosová rýchlosť pamätí RAM procesoru CPU je oveľa menšia (asi 10x) ako prenosová rýchlosť pamätí GPU. Dátový prenos pamäti GPU sa pohybuje na úrovni 140 GBps (GTX 280), resp. 177.4 GBps (GTX 480), zatiaľ čo prenosová rýchlosť DDR3 do 20 GBps [14]. V tomto ohľade má GPU oproti CPU navrch.

Tretí kernel slúži na presun vozidiel cez križovatku, kde sa algoritmus oveľa viac vetví než v ostatných dvoch kerneloch. Toto bude aj príčinou, prečo CPU trávi oveľa menej času v treťom kerneli oproti GPU. CPU je lepšie prispôsobené na vetviace sa algoritmy, obsahuje rôzne heuristiky a predikcie skokov a je to prispôsobené zariadenie pre všeobecné účely. V tabuľke príloh v Dodatku A si napríklad môžeme zobrať záznam všetkých troch zariadení s veľkosťou mapy, počtom vozidiel v mape 80534. Zobratím do úvahy iba prvé dva kenely, karta GTX 280 získa 11-násobný nárast výkonu a karta GTX 480 dosahuje 49.6-násobný nárast výkonu oproti CPU. Keď však berieme všetky 3 kernely ako celok, tieto hodnoty sa nám znížia na 4.39-násobný nárast výkonu pre GTX 280 a 21.16-násobný nárast výkonu karty GTX 480 oproti CPU.

Z časového hľadiska vykonáva CPU výpočet tretieho kernelu v priemere asi 5-krát rýchlejšie oproti grafickej karte GTX 280 a 2.5-krát rýchlejšie oproti karte GTX 480. Toto spôsobuje, že nárast výkonu grafických kariet je vďaka tretiemu kernelu znížený zhruba o polovicu v kerneloch s kvadrantmi s veľkým množstvom vozidiel.

zariadenie	počet vozidiel	veľkosť mapy	čas jedného behu v [ms]	FPS[1]
Core i7	5022	1024x1024	52.219	19.1501
GTX 280	5022	1024x1024	6.585	151.86
GTX 480	5022	1024x1024	2.606	383.73
Core i7	40124	1024x1024	55.803	17.9202
GTX 280	40124	1024x1024	17.267	57.9139
GTX 480	40124	1024x1024	4.108	243.427
Core i7	80625	1024x1024	64.817	15.4281
GTX 280	80625	1024x1024	21.441	46.6396
GTX 480	80625	1024x1024	6.02	166.113
Core i7	20277	2048x2048	226.89	4.40742
GTX 280	20277	2048x2048	26.255	38.088
GTX 480	20277	2048x2048	8.879	112.625
Core i7	161440	2048x2048	229.045	4.36595
GTX 280	161440	2048x2048	63.449	15.7606
GTX 480	161440	2048x2048	14.166	70.5916
Core i7	323685	2048x2048	271.419	3.68434
GTX 280	323685	2048x2048	80.624	12.4033
GTX 480	323685	2048x2048	21.813	45.8442

[1] FPS - Frames per second

Tabuľka 9.4: Čas trvania jedného behu trojice kernelov a FPS k ním prislúchajúce

9.3 Real-time zobrazovanie a predikcia stavu dopravnej situácie

Pre menšie mapy a menšie počty križovatiek pre účely real-time zobrazovania na výpočty s navrhnutým algoritmom postačí výkon procesoru CPU. Pokiaľ je však potreba mať výsledky simulácie v predstihu a dostatočnej rýchlosti, je nutné použiť zariadenie, ktoré má viac výpočtivých elementov. Problém simulácie cestnej premávky je s využitím celulárnych automatov pomerne dobre paralelizovateľný a dá sa dobre optimalizovať.

V nasledujúcej tabuľke je vybraných niekoľko konfigurácií a sú porovnané grafické karty GTX 280 a GTX 480 spolu s procesorom Intel i7 920. Vidieť, že s menšou mapou dosahuje CPU okolo 20 snímkov za sekundu (FPS). Keď zoberieme do úvahy, že na real-time zobrazovanie postačuje okolo 25 FPS, je 20 FPS dostatočná hodnota. Obe grafické karty vykazujú veľmi dobré výsledky pre real-time zobrazovanie a dali by sa využiť pre predikciu dopravnej situácie s až 15-násobným zrýchlením oproti behu v real-time.

Kapitola 10

Problémy pri implementácii a obmedzenia architektúry

Najväčším problémom programovania OpenCL kódu bolo ladenie chýb. Často sa stávalo, že program čítal alebo zapisoval na miesta, kde to nebolo chcené alebo dokonca mimo alokovanú a vyhradenú pamäť. Potom nastávali napríklad také situácie, že sa v premennej, určujúcej počet vozidiel v kvadrante, objavila hodnota väčšia, aká sa tu mala v skutočnosti nachádzať. Následne sa vlákno pokúšalo čítať dáta o vozidle z pamäti, ktorá buď neexistovala alebo nebola inicializovaná.

Pri chybnom prístupe sa do pamäti, beh kernelu neskončí chybovou hláškou, ako to je bežné pri klasickom programovaní. Kernel nijako neupozorní na chybný beh programu a programátor častokrát ani nevie, že niečo nie je v poriadku. Teda aspoň na prvý pohľad, kým si to nevšimne na výstupných dátach. Po zistení chybného správania programu si bolo postupne potrebné vypisovať časti dát, zužovať okruh problematického kódu až na (väčšinou) jeden konkrétny riadok, kde bol napríklad počítaný nesprávne index do poľa pamäti.

Samozrejme, na debug aplikácie by bolo možné použiť nástroj Nsight od spoločnosti NVIDIA, ktorý slúži na analýzu a ladenie GPGPU výpočtov s použitím CUDA C, OpenCL, DirectCompute, Direct3D a OpenGL [39]. Na ladenie je však nutné použiť 2 rôzne prepojené počítače, kde na jednom je spustená aplikácia a na druhom sa analyzuje a ladí kód. Túto možnosť nakoniec nebolo treba využiť a pre účely ladenia tejto aplikácie postačovalo vypisovanie na štandartný výstup konzoly.

Implementácia aplikácie prebiehala postupným optimalizovaním kódu kernelov predprípravou dát v krátkych častiach kódu, aby nebolo nutné vetviť väčšie časti kódu a tým tento kód zbytočne v rámci warpu serializovať. Ako ukázala kapitola 9.2, je dôležité, aby sa kód aplikácie vetvil čo najmenej, pretože vlákna v rámci jedného warpu, ktorý obsahuje 32 vlákien na testovaných grafických kartách, nemôžu naraz vykonávať viacero rôznych inštrukcií. Pokiaľ by malo mať každé vlákno rôznu vykonávanú cestu, beh každého vlákna sa serializuje a teda sa využíva iba jeden procesný element architektúry a tým ustupuje význam takéhoto multiprocesorového systému do úzadia.

Architektúra GPU je náchylná na výrazné spomalenie aplikácie, pokiaľ programátor nepoužíva zdieľanú pamäť tam, kde to je možné. Latencia zdieľanej pamäti je asi 100x menšia (viď sekcia 5.4.4) ako latencia globálnej. V implementovanej aplikácii cestnej premávky bola využívaná zdieľaná pamäť a priniesla badateľný nárast výkonu.

Keďže je problém urýchľovania simulácie dopravnej situácie pomocou GPGPU „dlhodobým“ procesom, kde pre odsimulovanie určitého časového okamžiku je nutné spustiť kernel

niekoľkokrát, nie je problémom dlhá inicializácia zariadenia GPU, ktorá trvá rádovo 100-200ms. Aplikácie, ktoré je možné paralelizovať, pričom nastane aj veľké zrýchlenie oproti CPU, ale ich beh trvá menej, ako je samotná inicializácia grafickej karty, je zbytočné takýmto spôsobom urýchľovať.

Medzi ďalšie (časové) obmedzenia architektúry a dobré praktiky patrí napríklad presun CPU kódu na GPU aj za cenu, že takýto presun neurýchli celkový výpočet aplikácie, ale zabráni prepnutiu kontextu z GPU do CPU a späť na GPU, pretože prepínanie kontextu takisto niečo stojí a je to lepšie sa mu vyhnúť, ak je to možné.

Kapitola 11

Rozšírenia aplikácie a budúci vývoj

Táto aplikácia má veľa možností a smerov, ktorými sa môže ďalej rozširovať a naberať na komplexnosti. V tejto kapitole budú popísané niektoré z nich.

Do ďalšieho vývoja by bolo možné zaradiť interoperabilitu medzi OpenCL a OpenGL. OpenGL sa v aplikácii používa pre grafické zobrazovanie simulácie. Momentálne sa po každom behu kernelu musia nakopírovať z globálnej pamäti GPU do pamäti CPU dáta reprezentujúce vozidlá pohybujúce sa po vozovkách. Ich zobrazovaním pomocou OpenGL sa opäť kopírujú späť do globálnej pamäti grafickej karty. Tomuto kroku by sa dalo pomocou interoperability OpenCL-OpenGL vyhnúť, pretože interoperabilita dáva možnosť priameho zobrazovania dát v globálnej pamäti grafickej karty, čo by pri zobrazovaní ušetrilo veľa času.

Z hľadiska GPU architektúry by bolo vhodné optimalizovať kernel počítajúci presun vozidiel cez križovatky. Tento kernel je pre grafickú kartu bottleneckom, pretože sa tu nachádza veľa vetvení a znižuje nárast výkonu oproti CPU v niektorých prípadoch až o polovicu.

Súčasťou architektúry GPU je aj textúrovacia pamäť, ktorá sa v podstate správa ako globálna pamäť s tým rozdielom, že je cache-ovaná. V mnohých prípadoch je teda čítanie z nej rýchlejšie (pokiaľ nie je čítanie náhodné, ale v 2D okolí). Textúrovaciu pamäť však nebolo možné použiť s dátami kvadrantov, pretože sa využívajú ako zdrojové, tak aj cieľové dáta a tieto sú pri každom behu medzi sebou prehodené, takže raz sa zapisuje do jedného poľa a z druhého sa číta a inokedy to je naopak. Textúrovacia pamäť slúži ako read-only pamäť. Dala by sa však použiť pre uloženie rozloženia mapy, pretože z tejto pamäti sa v jednotlivých kerneloch iba číta.

Implementovaná aplikácia používa časový krok simulátora medzi výpočtami 1 sekunda a minimálná možná prejdená vzdialenosť áut je 5 metrov (veľkosť jednej bunky celulárneho automatu). Aby bolo možné presnejšie simulovať cestnú situáciu, je nutné simulátor modifikovať tak, aby bol schopný prevádzať menšie časové kroky medzi dvomi stavmi celulárneho automatu. To by však znamenalo, že by sa musela zmeniť aj reprezentácia vozidiel v pamäti tak, aby mohlo jedno auto zasahovať do viac ako jednej bunky súčasne pomocou napríklad percentuálneho rozdelenia v bunkách. Potom by už bolo jednoduché rozšíriť simulátor o rôzne typy vozidiel, napr. osobné, nákladné, autobus a pod.

Kapitola 12

Záver

Táto práca sa zaoberá témou akcelerácia mikroskopickej simulácie dopravy za použitia OpenCL. Dôvody a motivácie, prečo je nutné sa takýmito témami zaoberať, sú popísané v prvej úvodnej kapitole. Ide hlavne o problémy spojené s pribúdajúcim množstvom vozidiel na našich cestách a s tým spojené častejšie havárie, väčšie zápchy a zvýšenie vypúšťaných emisií CO2, ktoré znečisťujú životné prostredie. Ďalej sa v tejto kapitole rozoberajú niektoré najdôležitejšie spôsoby riešenia týchto problémov.

V nasledujúcich kapitolách sa popisujú technológie spojené s ďalším pokračovaním práce. Dôležitou súčasťou je základný systém, na ktorom bude simulácia postavená - celulárne automaty. Vďaka tomu, že ide o diskrétny výpočtový paralelný model s lokálnou interakciou výpočtových modelov, je možné problém simulácie dopravy naprogramovať paralelne tak, aby dochádzalo na systémoch s viacerými procesormi k zrýchleniu simulácie.

Architektúrou použitou pri implementácii práce je OpenCL s technikou GPGPU, ktorú nám poskytujú dnešní výrobcovia grafických kariet. Grafické karty sú vysokovýkonné mnohojadrové procesory s vysokým výpočtovým potenciálom a dátovou priepustnosťou, použiteľné na všeobecné výpočty. Tento princíp sa hodí hlavne z dôvodu, že mikroskopický model dopravy je možné simulovať paralelne, každé vozidlo nezávisle na ostatných.

Okrem návrhu a implementácie mikroskopického simulátora s použitím OpenCL bolo úlohou práce otestovať výkonnosť simulátora s ohľadom na rôzne parametre celulárneho automatu, ako aj diskusia ďalších úprav, rozšírení a nutných obmedzení simulátora s ohľadom na architektúru GPU.

Implementovanou aplikáciou bolo dosiahnuté navýšenie rýchlosti spracovania krokov simulátora na grafických kartách GPU oproti procesorom CPU, a to niekoľkonásobne. Testy porovnávajú rýchlosť výpočtov na GPU, konkrétne NVIDIA GeForce GTX 280 a NVIDIA GeForce GTX 480 oproti štvorjadrovému procesoru CPU Intel Core i7 920. Testy na CPU boli prevádzané s využitím všetkých jadier CPU vďaka ovládaču OpenCL firmy Intel.

Program OpenCL sa skladá z troch kernelov, z ktorých každý má inú úlohu. Prvý spúšťaný kernel z trojice kernelov sa stará o zistenie novej rýchlosti vozidiel, ako aj o ich posun po mape (nastavenie novej pozície). Druhý kernel slúži iba na kopírovanie dát z dočasnej globálnej pamäti, ktoré pripravil prvý kernel, do globálnej pamäti kvadrantov a tretí sa zaoberá prevádzaním vozidiel cez križovatky s kontrolou, aby odbočujúce autá nekrížili cestu vozidlám, ktoré sa nachádzajú na hlavnej ceste, príp. majú prednosť.

Keď zoberieme napríklad konfiguráciu aplikácie nasledovnú: veľkosť mapy 1024x1024 buniek, počet vozidiel 40124, vzdialenosť medzi križovatkami 200 metrov, bolo dosiahnuté zrýchlenie oproti procesoru CPU 3.23-násobné na grafickej karte GeForce GTX 280 a 13.23-násobné na grafickej karte GeForce GTX 480. Pri bližšej analýze výsledkov testov bolo

zistené, že tretí kernel znižuje navýšenie výkonu oproti CPU v niektorých prípadoch až na polovicu. Je to spôsobené tým, že tretí kernel obsahuje veľa podmienok, musí sa často vetviť a tým sa kód kernelu musí serializovať. Keby sme tretí kernel z výpočtov vypustili, tak pre vyššie uvedenú konfiguráciu by bola karta GTX 280 rýchlejšia oproti CPU 7.9-násobne a karta GTX 480 rýchlejšia 29.28-násobne.

Ďalej bolo zistené, že CPU trávi veľa času vykonávaním druhého kernelu, a to v priemere 50% z celkového času behu všetkých troch kernelov. Grafické karty tento kernel počítajú oveľa rýchlejšie a ich strávený čas v tomto kerneli sa pohybuje v priemere na 7% z celkového času behu všetkých troch kernelov. Tento fenomém môže byť spôsobený tým, že prenosová rýchlosť globálnej pamäti grafickej karty GTX 280 je 140 GBps a 177.4 GBps karty GTX 480, zatiaľ čo prenosová rýchlosť pamäťi DDR3 procesoru Intel Core i7 je oveľa nižšia, a to do 20 GBps. Medzi ďalšie faktory, ktoré môžu spôsobovať takéto správanie, by sme mohli zaradiť ovládače OpenCL firmy Intel vo verzii alpha, prípadne ďalšiu réžiu procesoru CPU.

V kapitole 11 sú popísané niektoré ďalšie rozšírenia simulátora a budúci vývoj. V prvom rade sem môžeme zaradiť lepšie grafické spracovanie a využitie interoperability OpenCL-OpenGL pre rýchlejšie zobrazovanie grafického výstupu. Pre rýchlejší výpočet kernelov je možné optimalizovať OpenCL kód tak, aby tretí kernel bol vykonávaný rýchlejšie a aby ho warpy nemuseli až tak serializovať. Z hľadiska presnosti simulátora by bolo možné zmeniť časový krok aplikácie a tým získať podrobnejšie výsledky simulácie. Pre dosiahnutie ešte väčšieho priblíženia sa realite môžu byť do simulátora pridané rôzne typy áut (rôzna dĺžka, maximálna rýchlosť, ...).

Literatúra

[1] Benjamin, S. C.; Johnson, N. F.; Hui, P. M.: *Cellular automata models of traffic flow along a highway containing a junction.* J. Phys. A: Math. Gen. 29 3119, 1996.

[2] Komenda, T.: *Sebereplikace v celulárních systémech.* Brno, FIT VUT v Brně, 2009.

[3] Korček, P.; Sekanina, L.; Fučík, O.: *Towards Scalable and Accurate Microscopic Traffic Simulation Using Advanced Cellular Automata Based Models.* IEEE Intelligent Transportation Systems Society, 2010.

[4] Lee, D.-H.; Chandrasekar, P.: *A framework for parallel traffic simulation using multiple instancing of a simulation program.* Department of Civil Engineering, National University of Singapore.

[5] Nagel, K.; Schreckenberg, M.: *A cellular automaton model for freeway traffic.* Journal de Physique I 2, 1992.

[6] NVIDIA: *NVIDIA CUDA C Programming Guide.* NVIDIA Corporation, 2010.

[7] NVIDIA: *OpenCL Programming Guide for the CUDA Architecture.* NVIDIA Corporation, 2010.

[8] Sekanina, L.: *Development v evolučním návrhu.* Brno, FIT VUT v Brně, 2010.

[9] Strippgen, D.; Nagel, K.: *Multi-Agent Traffic Simulation with CUDA*. High Performance Computing & Simulation, 2009.

[10] Wolfram, S.: *Cellular Automata.* 1983.

[11] Wolfram, S.: *Statistical Mechanics of Cellular Automata.* 1983.

[12] WWW stránky: About GPGPU.org. `http://gpgpu.org/about`.

[13] WWW stránky: Avivo vs. Purevideo. `http://www.tomshardware.com/reviews/avivo-vs-purevideo,1492.html`.

[14] WWW stránky: Benchmark Results: Overclocking, Latency, And Bandwidth. `http://www.tomshardware.com/reviews/8gb-ddr3-ram,2542-8.html`.

[15] WWW stránky: CineFX (NV30) Inside. `http://alt.3dcenter.org/artikel/cinefx/index_e.php`.

[16] WWW stránky: Confessions of a Speed Junkie. `http://gpgpu-computing.blogspot.com/`.

[17] WWW stránky: The Facts About Hybrid Car Emissions and Global Warming.
http://www.carbon-monoxide-poisoning.com/article5-hybrid-car-emissions.html.

[18] WWW stránky: Flynn's taxonomy.
http://en.wikipedia.org/wiki/Flynn%27s_taxonomy.

[19] WWW stránky: Fuel-efficient Driving.
http://eartheasy.com/move_fuel_efficient_driving.html.

[20] WWW stránky: Fuel Saving and minimizing CO2 emissions.
http://www.zf.com/corporate/en/products/innovations/8hp_automatic_
transmissions/lower_consumption/lower_consumption.html.

[21] WWW stránky: GeForce 8800. http://www.nvidia.com/page/geforce_8800.html.

[22] WWW stránky: GeForce FX. http://www.nvidia.com/page/fx_desktop.html.

[23] WWW stránky: GeForce MX. http://www.nvidia.com/page/geforce4.html.

[24] WWW stránky: GeForce3 Product Overview.
http://www.nvidia.com/object/LO_20010612_4376.html.

[25] WWW stránky: Intellisample 4.0.
http://www.nvidia.com/object/feature_intellisample4.0.html.

[26] WWW stránky: Introduction to Micro-simulation.
http://www.microsimulation.drfox.org.uk/intro.html.

[27] WWW stránky: Leadtek WinFast GeForce 256 DDR Review.
http://www.tomshardware.co.uk/leadtek-winfast
-geforce-256-ddr-review,review-152.html.

[28] WWW stránky: Metropolitan Road Traffic Simulation on FPGAs.
http://www.rasr.lanl.gov/Apps/docs/transim_fccm05.pdf.

[29] WWW stránky: Microsimulation.
http://www.statcan.gc.ca/microsimulation/index-eng.htm.

[30] WWW stránky: Ministerstvo dopravy, výstavby a regionálneho rozvoja Slovenskej republiky - Dopravná infraštruktúra.
http://www.telecom.gov.sk/files/statistika_vud/dop_infra.htm.

[31] WWW stránky: Ministerstvo dopravy, výstavby a regionálneho rozvoja Slovenskej republiky - Dopravné prostriedky.
http://www.telecom.gov.sk/files/statistika_vud/dop_prostriedky.htm.

[32] WWW stránky: Ministerstvo dopravy, výstavby a regionálneho rozvoja Slovenskej republiky - Preprava - osobná preprava.
http://www.telecom.gov.sk/files/statistika_vud/preprava_osob.htm.

[33] WWW stránky: NV1 Specs.
http://nvidiagraphicscards.com/nvidia-nv1-information.php.

[34] WWW stránky: NVIDIA GEFORCE 6 SERIES SPECIFICATIONS. http://http.download.nvidia.com/ndemand/product_overview/PO_GeForce_6_series_24.pdf.

[35] WWW stránky: NVIDIA GeForce 8800 GPU Architecture Overview. http://www.nvidia.com/object/IO_37100.html.

[36] WWW stránky: NVIDIA GeForce FX 5900 Ultra 256MB Video Card Review. http://ixbtlabs.com/articles2/gffx/5900u.html.

[37] WWW stránky: Nvidia GeForce GTX 200 Graphics Architecture Review: Born to Win? http://www.xbitlabs.com/articles/graphics/display/geforce-gtx200-theory_10.html.

[38] WWW stránky: NVIDIA Introduces GeForce FX (NV30). http://www.anandtech.com/show/1034.

[39] WWW stránky: NVIDIA Parallel Nsight. http://developer.nvidia.com/nvidia-parallel-nsight.

[40] WWW stránky: NVIDIA Quadro, Exponentially better for Autocad. http://www.nvidia.com/docs/IO/40049/NV_AutoCAD_Nov10_LR.pdf.

[41] WWW stránky: NVIDIA RIVA TNT. http://www.tomshardware.com/reviews/3d-chips,83-3.html.

[42] WWW stránky: NVIDIA SLI ZONE. http://www.slizone.com/page/home.html.

[43] WWW stránky: NVIDIA's Double Graphics Whopper: SLI Comes to Market. http://www.tomshardware.com/reviews/nvidia,930.html.

[44] WWW stránky: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

[45] WWW stránky: OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/.

[46] WWW stránky: OpenCL Optimization. http://www.nvidia.com/content/GTC/documents/1068_GTC09.pdf.

[47] WWW stránky: OpenGL Overview. http://www.opengl.org/about/overview/.

[48] WWW stránky: Public Transport Vs Private Transport. http://www.energysavingsecrets.co.uk/PublicTransportVsPrivateTransportTheDebate.html.

[49] WWW stránky: Quadstone Paramics. http://www.paramics-online.com/.

[50] WWW stránky: RESEARCH ANALYSIS: Review of stop-start systems. http://www.just-auto.com/analysis/review-of-stop-start-systems_id101657.aspx.

[51] WWW stránky: Ročenka dopravy České republiky. https://www.sydos.cz/cs/rocenka_pdf/Rocenka_dopravy_2009.pdf.

[52] Švéda, P.: *Simulátor dopravy na pozemních komunikacích.* Brno, FIT VUT v Brně, 2010.

Dodatok A

Namerané výsledky

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovatkami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
GTX 280	1024	5022	200	2.729	0.317	3.54	6.585
GTX 280	1024	3490	300	2.651	0.333	2.535	5.519
GTX 280	1024	2187	500	2.312	0.284	0.968	3.565
GTX 280	1024	1387	800	2.181	0.301	0.466	2.948
GTX 280	1024	20214	200	5.467	0.515	8.991	14.973
GTX 280	1024	14126	300	4.451	0.405	4.03	8.886
GTX 280	1024	8787	500	3.597	0.385	1.677	5.659
GTX 280	1024	5559	800	3.373	0.351	0.666	4.389
GTX 280	1024	40124	200	6.531	0.481	10.255	17.267
GTX 280	1024	28088	300	5.564	0.464	4.95	10.979
GTX 280	1024	17465	500	5.125	0.425	2.338	7.887
GTX 280	1024	11113	800	4.505	0.409	0.775	5.688
GTX 280	1024	60388	200	6.9	0.509	11.236	18.645
GTX 280	1024	42500	300	6.134	0.481	5.516	12.132
GTX 280	1024	26304	500	5.948	0.468	2.474	8.89
GTX 280	1024	16771	800	4.972	0.437	0.845	6.254
GTX 280	1024	80625	200	7.788	0.588	13.064	21.441
GTX 280	1024	56765	300	7.08	0.555	6.409	14.044
GTX 280	1024	35177	500	5.871	0.457	2.77	9.098
GTX 280	1024	22532	800	5.323	0.457	0.916	6.697
GTX 280	2048	20277	200	10.669	0.931	14.655	26.255
GTX 280	2048	13840	300	9.828	0.906	7.816	18.55
GTX 280	2048	8501	500	8.579	0.856	2.995	12.429
GTX 280	2048	5250	800	7.814	0.842	1.393	10.049
GTX 280	2048	80534	200	19.287	1.258	32.139	52.684
GTX 280	2048	54918	300	16.714	1.224	14.175	32.113
GTX 280	2048	33542	500	14.294	1.087	4.798	20.187

Tabuľka A.1: Dosiahnuté výsledky - GTX 280

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovat-kami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
GTX 280	2048	21046	800	12.775	1.033	1.708	15.517
GTX 280	2048	161440	200	23.422	1.357	38.67	63.449
GTX 280	2048	110318	300	21.526	1.324	17.579	40.429
GTX 280	2048	67142	500	20.046	1.382	7.234	28.663
GTX 280	2048	41776	800	16.938	1.298	2.329	20.565
GTX 280	2048	242495	200	25.612	1.654	42.75	70.016
GTX 280	2048	165736	300	23.022	1.534	18.79	43.346
GTX 280	2048	100983	500	21.357	1.711	7.464	30.532
GTX 280	2048	62876	800	19.547	1.447	2.878	32.873
GTX 280	2048	323685	200	29.449	1.663	49.513	80.624
GTX 280	2048	221330	300	26.427	1.894	21.841	50.162
GTX 280	2048	134917	500	22.675	1.394	8.316	32.385
GTX 280	2048	83887	800	20.421	1.528	3.117	25.066

Tabuľka A.2: Dosiahnuté výsledky - GTX 280

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovat-kami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
GTX 480	1024	5022	200	0.946	0.199	1.461	2.606
GTX 480	1024	3490	300	0.658	0.193	0.912	1.763
GTX 480	1024	2187	500	0.547	0.168	0.547	1.263
GTX 480	1024	1387	800	0.433	0.163	0.367	0.974
GTX 480	1024	20214	200	1.215	0.211	1.853	3.278
GTX 480	1024	14126	300	0.953	0.199	1.033	2.185
GTX 480	1024	8787	500	0.724	0.202	0.543	1.468
GTX 480	1024	5559	800	0.577	0.158	0.403	1.138
GTX 480	1024	40124	200	1.66	0.212	2.236	4.108
GTX 480	1024	28088	300	1.258	0.212	1.248	2.717
GTX 480	1024	17465	500	0.912	0.19	0.61	1.711
GTX 480	1024	11113	800	0.702	0.159	0.453	1.314
GTX 480	1024	60388	200	2.225	0.23	2.636	5.091
GTX 480	1024	42500	300	1.653	0.218	1.36	3.232
GTX 480	1024	26304	500	1.115	0.222	0.767	2.104
GTX 480	1024	16771	800	0.868	0.201	0.515	1.585
GTX 480	1024	80625	200	2.74	0.249	3.032	6.02
GTX 480	1024	56765	300	2.044	0.245	1.577	3.866
GTX 480	1024	35177	500	1.405	0.22	0.96	2.586
GTX 480	1024	22532	800	0.976	0.196	0.522	1.694

Tabuľka A.3: Dosiahnuté výsledky - GTX 480

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovat-kami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
GTX 480	2048	20277	200	3.195	0.392	5.292	8.879
GTX 480	2048	13840	300	2.32	0.374	2.651	5.345
GTX 480	2048	8501	500	1.701	0.353	1.087	3.141
GTX 480	2048	5250	800	1.291	0.364	0.648	2.303
GTX 480	2048	80534	200	4.177	0.419	6.328	10.923
GTX 480	2048	54918	300	3.437	0.407	3.102	6.946
GTX 480	2048	33542	500	2.515	0.389	1.415	4.318
GTX 480	2048	21046	800	1.78	0.385	0.713	2.878
GTX 480	2048	161440	200	5.893	0.444	7.829	14.166
GTX 480	2048	110318	300	4.195	0.413	3.631	8.239
GTX 480	2048	67142	500	2.885	0.385	1.607	4.876
GTX 480	2048	41776	800	2.228	0.394	0.818	3.439
GTX 480	2048	242495	200	8.086	0.491	9.577	18.154
GTX 480	2048	165736	300	5.792	0.452	4.413	10.657
GTX 480	2048	100983	500	3.782	0.421	1.849	6.052
GTX 480	2048	62876	800	2.635	0.397	0.888	3.92
GTX 480	2048	323685	200	10.03	0.506	11.277	21.813
GTX 480	2048	221330	300	7.152	0.483	5.146	12.781
GTX 480	2048	134917	500	4.624	0.425	2.048	7.098
GTX 480	2048	83887	800	3.096	0.421	0.948	4.464

Tabuľka A.4: Dosiahnuté výsledky - GTX 480

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovat-kami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
i7 920	1024	5022	200	21.982	29.529	0.708	52.219
i7 920	1024	3490	300	22.19	30.644	0.357	53.191
i7 920	1024	2187	500	20.283	28.488	0.186	48.958
i7 920	1024	1387	800	20.15	28.435	0.107	48.693
i7 920	1024	20214	200	22.433	28.514	0.809	51.737
i7 920	1024	14126	300	21.36	28.514	0.449	50.323
i7 920	1024	8787	500	20.839	28.335	0.219	49.393
i7 920	1024	5559	800	20.385	28.224	0.119	48.728
i7 920	1024	40124	200	25.884	28.927	0.992	55.803
i7 920	1024	28088	300	25.294	31.681	0.533	57.508
i7 920	1024	17465	500	22.476	29.452	0.255	52.183
i7 920	1024	11113	800	20.92	28.25	0.124	29.294
i7 920	1024	60388	200	29.655	28.84	1.222	59.717
i7 920	1024	42500	300	27.81	30.685	0.652	59.147
i7 920	1024	26304	500	23.487	28.522	0.301	52.309

Tabuľka A.5: Dosiahnuté výsledky - Intel Core i7

Zariadenie	Veľkosť mapy	Počet áut	Vzdialenosť medzi križovat-kami	Kernel1 [ms]	Kernel2 [ms]	Kernel3 [ms]	Spolu [ms]
i7 920	1024	16771	800	23.225	29.631	0.136	52.992
i7 920	1024	80625	200	35.018	28.366	1.433	64.817
i7 920	1024	56765	300	29.464	28.776	0.739	58.979
i7 920	1024	35177	500	25.208	28.603	0.334	54.146
i7 920	1024	22532	800	24.272	30.245	0.143	54.661
i7 920	2048	20277	200	95.261	129.106	2.523	226.89
i7 920	2048	13840	300	92.015	124.21	1.224	217.449
i7 920	2048	8501	500	86.788	120.313	0.529	207.63
i7 920	2048	5250	800	84.879	119.864	0.229	204.972
i7 920	2048	80534	200	101.098	126.929	3.05	231.078
i7 920	2048	54918	300	91.965	119.225	1.566	212.785
i7 920	2048	33542	500	89.275	119.356	0.692	209.323
i7 920	2048	21046	800	87.891	119.097	0.269	207.258
i7 920	2048	161440	200	107.262	117.88	3.903	229.045
i7 920	2048	110318	300	100.809	119.659	1.887	222.355
i7 920	2048	67142	500	94.205	119.813	0.802	214.82
i7 920	2048	41776	800	92.058	119.971	0.3	212.329
i7 920	2048	242495	200	125.831	118.253	5.029	249.113
i7 920	2048	165736	300	110.976	119.255	2.438	232.668
i7 920	2048	100983	500	100.849	119.138	1.006	220.993
i7 920	2048	62876	800	96.437	120.822	0.35	217.609
i7 920	2048	323685	200	147.099	118.469	5.852	271.419
i7 920	2048	221330	300	124.361	119.472	2.813	246.646
i7 920	2048	134917	500	111.008	123.565	1.116	235.689
i7 920	2048	83887	800	97.923	119.536	0.359	217.819

Tabuľka A.6: Dosiahnuté výsledky - Intel Core i7

Dodatok B

Manuál k aplikácii

Aplikáciu je možné preložiť pomocou vývojového prostredia Visual Studio 2010, ktorého projekt je súčasťou priloženého CD média. Aby preklad prebehol v poriadku, je nutné mať nainštalované ovládače Nvidia verzie 270.61 alebo vyššie. Ďalej je nutné mať nainštalovaný OpenCL ovládač firmy Intel pre možnosť spúšťania OpenCL kódu na procesore CPU. Keďže v dobe písania tejto práce vyšiel nový ovládač OpenCL firmy Intel verzie beta a práca nebola s týmto ovládačom testovaná, ovládač verzie alpha je priložený na CD médiu.

Po úspešnom preložení aplikácie vo Visual Studiu 2010 sa dá s modelom experimentovať pomocou argumentov príkazového riadku nasledovne:

parameter	funkcia parametra a možnosti
-help	Zobrazenie pomocnej obrazovky
-command n	Tento parameter určuje, či sa bude zobrazovať grafický výstup aplikácie alebo nie možnosti: 1 - zobraziť grafiku (prednastavená hodnota) 2 - nezobraziť grafiku (pre účely testovania, každých 1000 behov sa zobrazia na štandartný výstup štatistiky - priemerné hodnoty behov kernelov)
-platform n	Platforma, na ktorej sa bude počítať OpenCL kód možnosti: 1 - GPU (prednastavená hodnota) 2 - CPU
-size n	Veľkosť mapy možnosti: 1 - 1024x1024 buniek (prednastavená hodnota) 2 - 2048x2048 buniek
-density f	Hustota cestnej premávky v rozsahu <0.05, 0.8> (prednastevená hodnota: 0.2 predstavuje 20%)
-distance n	Vzdialenosť medzi križovatkami v metroch (prednastavená hodnota je 250)
-repeat n	Počet opakovaní behu kernelov, využíva sa iba pri nezobrazovaní grafiky (prednastavená hodnota je 2000)
-crossroad n	Parameter určuje, ako často (po koľkých behoch kernelov) sa majú meniť pravdepodobnosti križovatiek pre odbočenie vozidiel (prednastavená hodnota je 60)

Tabuľka B.1: Možnosti argumentov pri spúšťaní aplikácie

úkon myšou	funkcia
pravý klik + posúvanie myši hore / dole	zoom in / out
ľavý klik + posúvanie myši žiadaným smerom	posum po mape

Tabuľka B.2: Ovládanie spustenej aplikácie s grafickým zobrazením pomocou myši

klávesová skratka	funkcia
'+'	Zvýšenie framerate-u (počtu kernelov za sekundu)
'-'	Zníženie framerate-u (počtu kernelov za sekundu)
medzera	Pozastavenie / spustenie animácie
'i'	Informačný panel
Esc / 'q' / 'x'	Skončenie aplikácie

Tabuľka B.3: Ovládanie spustenej aplikácie s grafickým rozhraním pomocou klávesnice

Limitácie implementácie:

- vzdialenosť medzi križovatkami musí byť násobkom čísla 5 (predstavuje veľkosť bunky)
- minimálna vzdialenosť medzi križovatkami je 200 metrov

Všetky argumenty sú voliteľné, každý má prednastavanú hodnotu.
Príklad spustenia aplikácie:

```
program.exe -platform 2 -size 2 -density 0.4 -distance 800
```

spustí program s výpočtom OpenCL kódu na CPU, veľkosťou mapy 2048x2048, hustotou premávky 40% a vzdialenosťou medzi križovatkami 800 metrov.